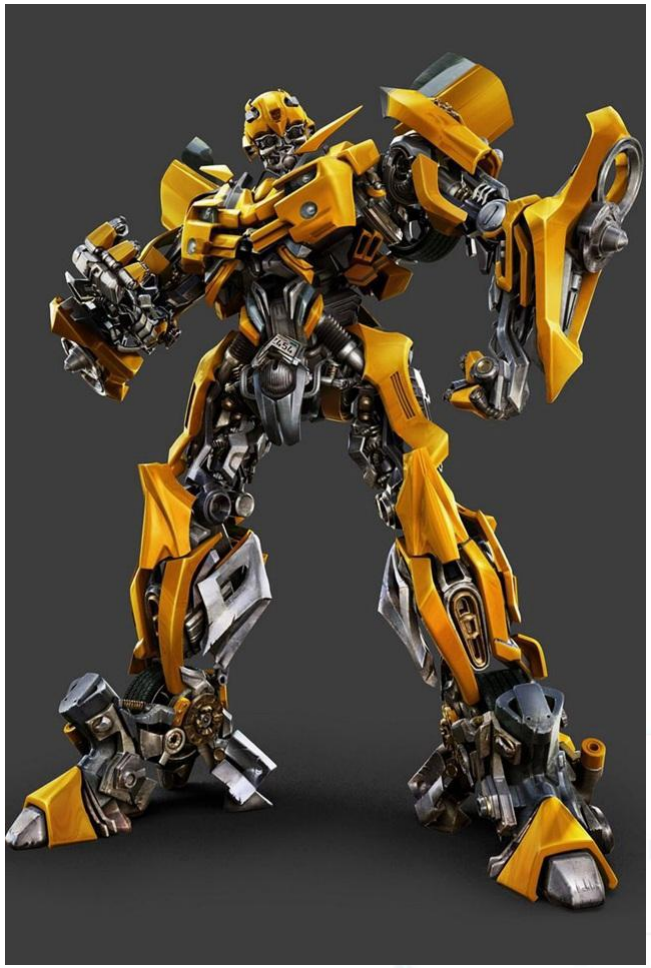
Transformer

汇报人：扣扣

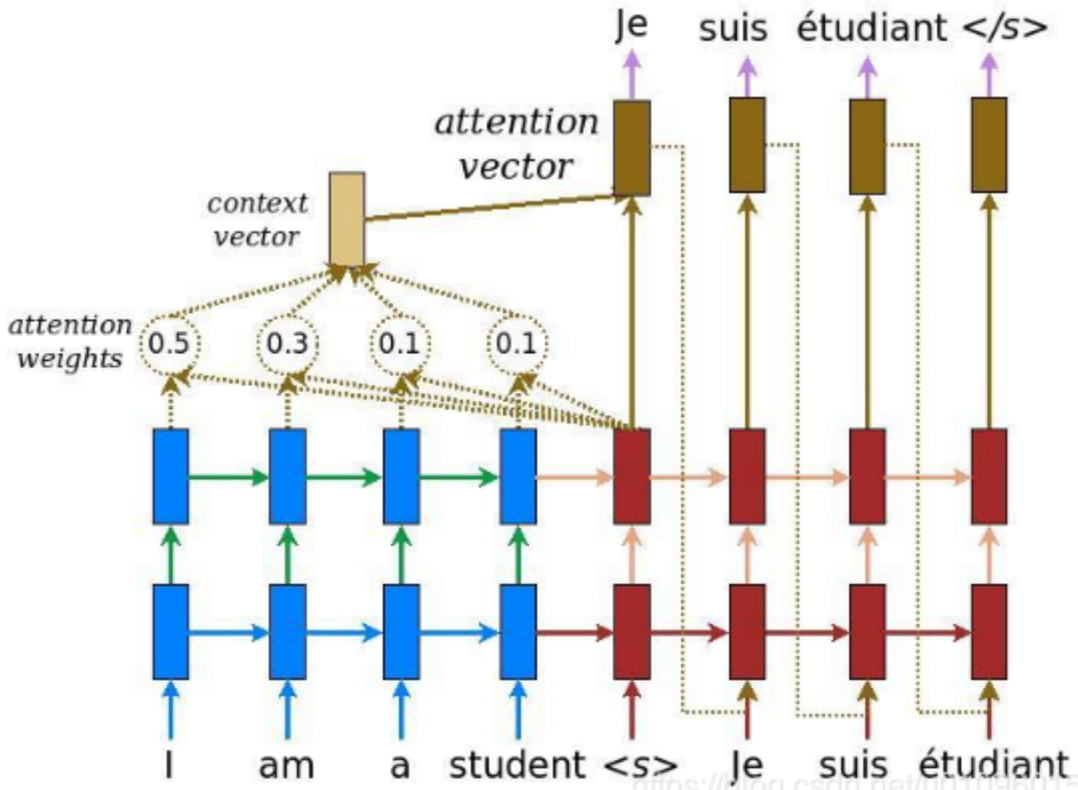
大纲

- 注意力机制在NLP上的发展历史
- Transformer详解
- Transformer扩展
  - Synthesizer: Rethinking Self-Attention in Transformer Models
  - Low-Rank Bottleneck in Multi-head Attention Models
  - Reformer



注意力机制在NLP上的发展历史

seq2seq+attention
RNN+RNN+attention

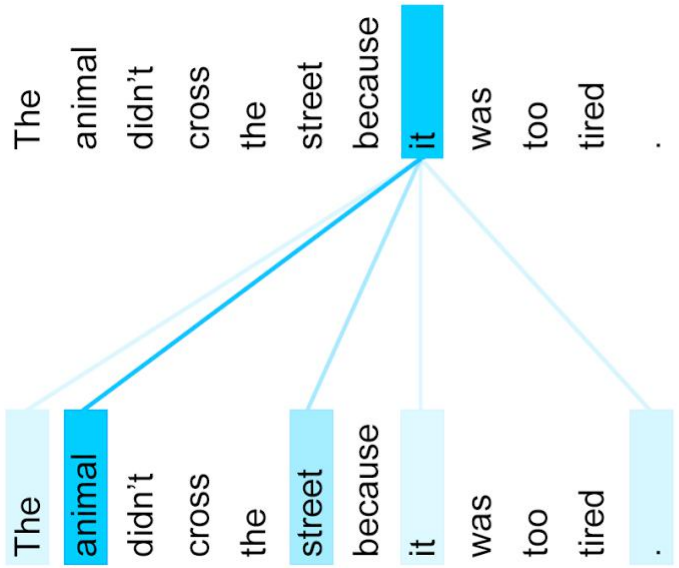


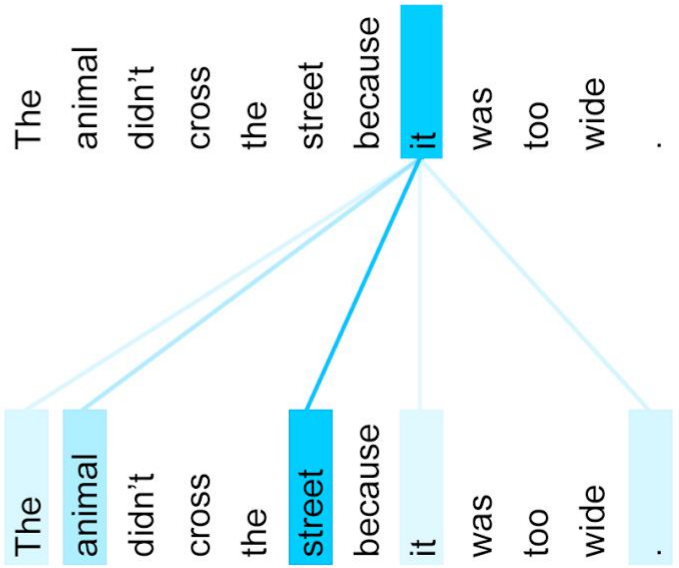
机器翻译：

- 输入与输出分别是两个序列
- 应用两个循环神经网络结构，一个用于对输入语句的编码，简称编码器（Encoder），另一个用于生成输出语句，简称解码器（Decoder）。
- 类比传统翻译过程中的对齐原理（源文与译文的词语存在对应性），编码过程中对输入语句中各词施加不同程度的注意力提高了翻译准确度。

注意力机制在NLP上的发展历史

self-attention

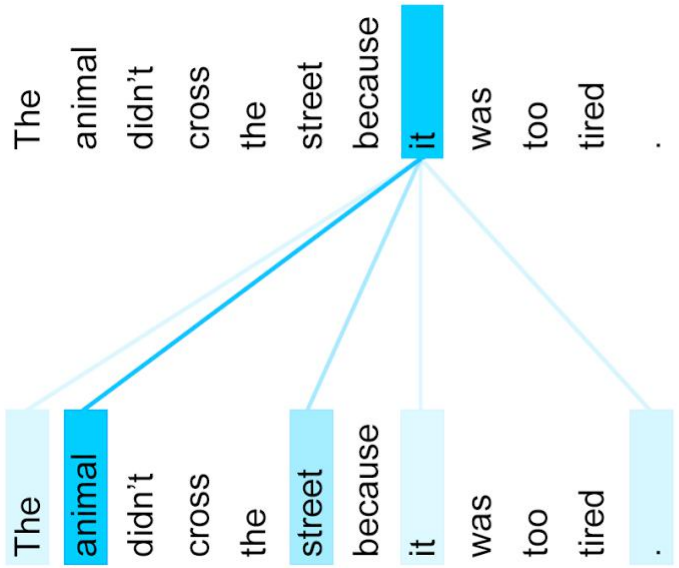


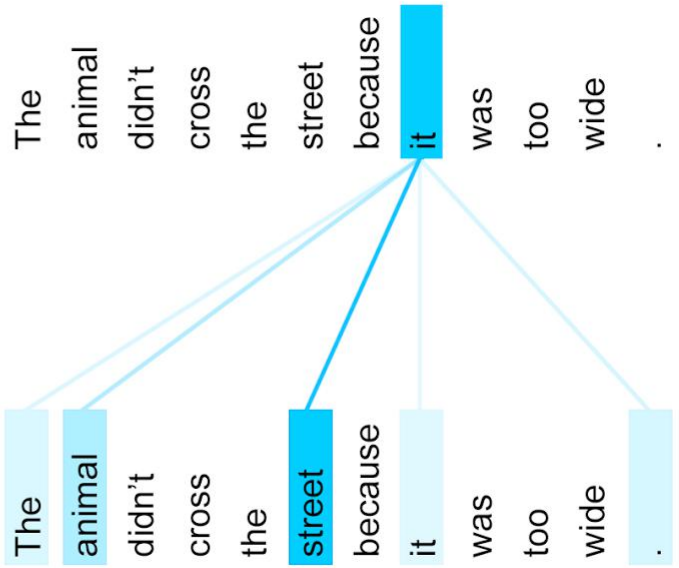


2017年，谷歌机器翻译团队完全摒弃了循环神经网络结构，提出的完全基于自注意力机制的Transformer模型在机器翻译任务上又取得了新的进展。

自注意力机制：句子对其本身的注意力，相当于源语句和目标语句为同一句子。

注意力机制在NLP上的发展历史





“the animal didn't cross the street because it was too wide”
句子中的“it”指的是“street”还是“animal”？

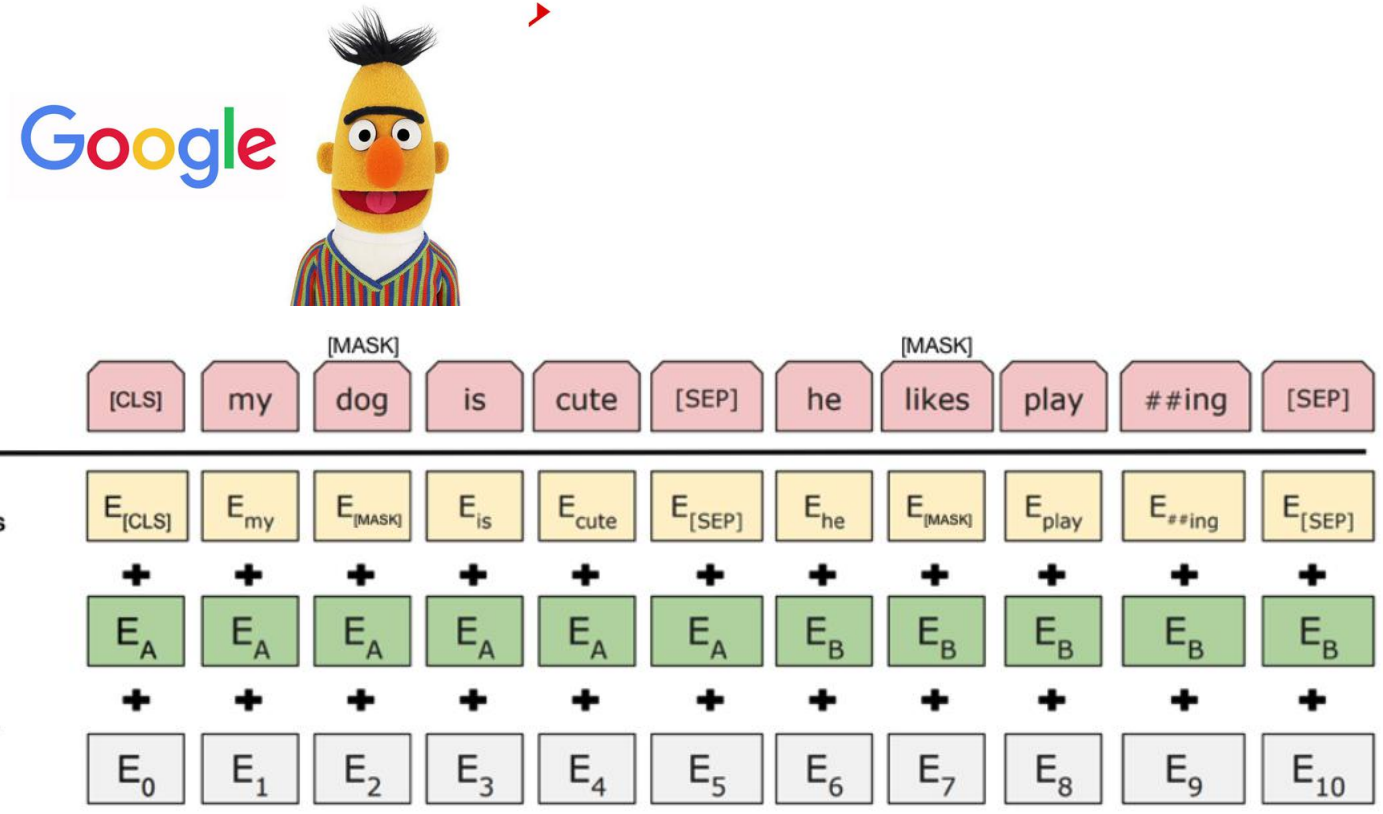
利用了自注意力模型的编码器，可以自主学习出同一句话中词与词之间的联系，因此获得更多更深层次的编码信息。
另外，由于没有使用循环神经网络结构，Transformer对于长句的处理能力以及运算速度也大幅提升。

注意力机制在NLP上的发展历史

Bert

2018年末，谷歌基于双向Transformer推出的Bert（Bidirectional Encoder Representations from Transformers）模型在11项自然语言处理任务中都刷新了记录，具备里程碑式的意义。

注意力机制在NLP上的发展历史

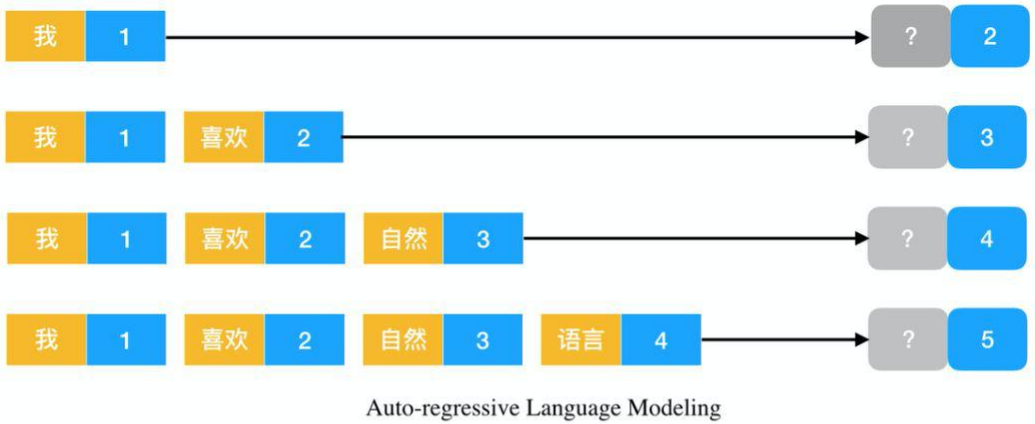


Bert训练数据：包含两个句子，有一半的可能性为上下文，并且随机遮盖住15%的词汇，让输出预测是否为上下文以及这些词汇。
结构：应用Transformer的编码过程。

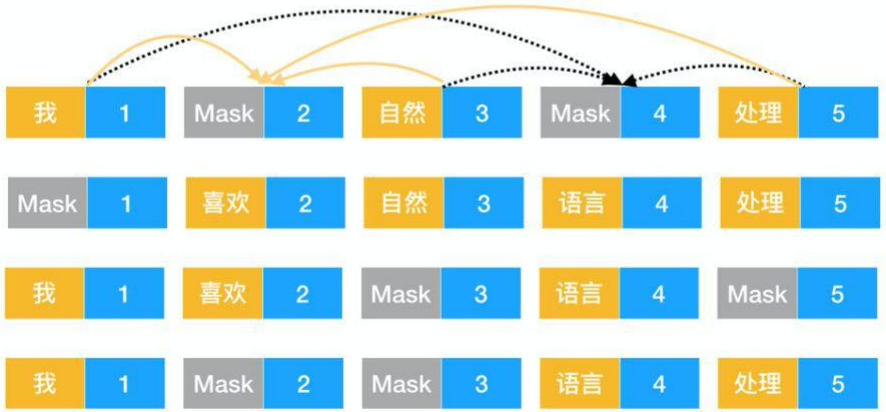
Bert是一个强大的预训练模型，只需要在某些特定层上稍加改造，便能够在许多下游任务上取得不错的效果。

预训练+微调模式

注意力机制在NLP上的发展历史



Auto-regressive Language Modeling

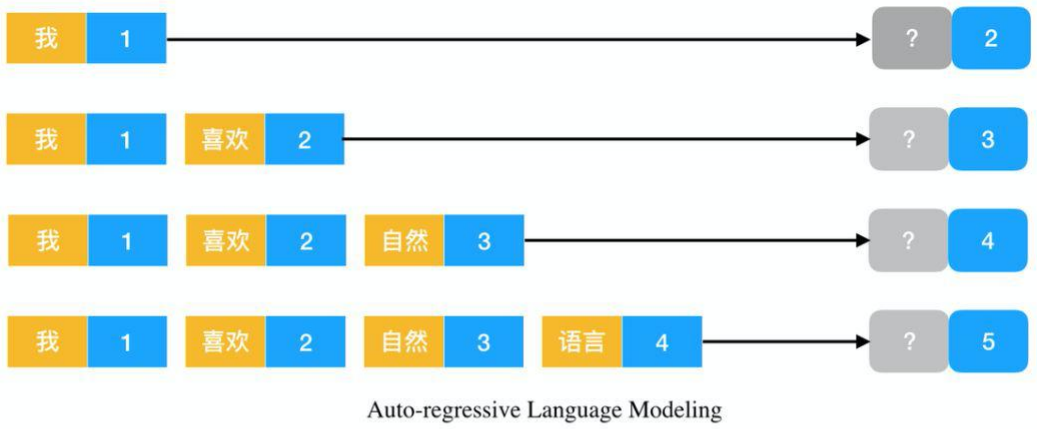


Auto-encoding Language Modeling

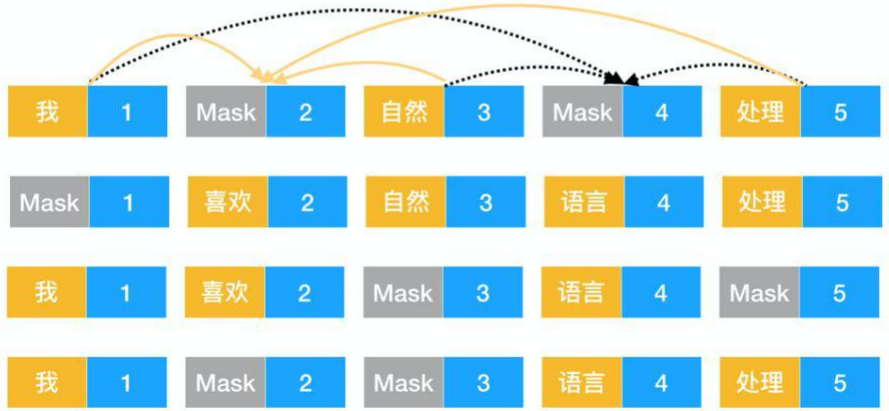
2019年初，谷歌和卡耐基梅隆大学联合推出的XLNet在20个任务上超越了Bert，并且在18项任务上取得了历史最优的成绩。

遮盖的弊端：训练过程中需要遮盖部分词汇作为输入，而在下游任务的微调过程中并不需要遮盖，因此存在训练与微调过程数据形式不一致的问题，会减弱模型性能。

注意力机制在NLP上的发展历史



Auto-regressive Language Modeling



Auto-encoding Language Modeling

XLNet：

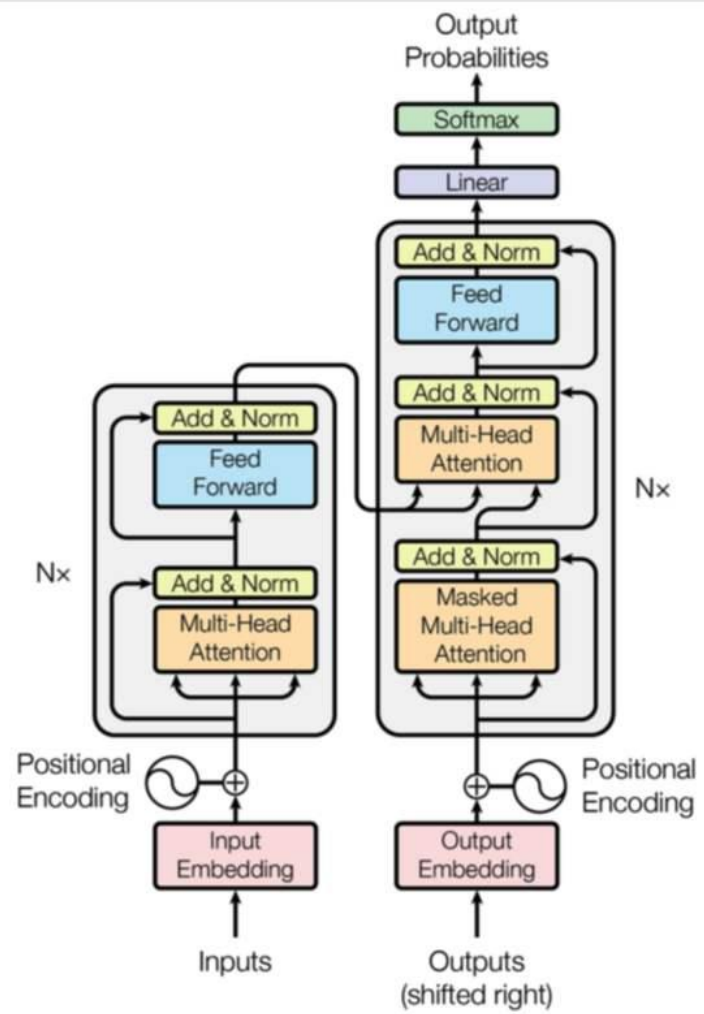
- 在自回归语言模型的基础上，摒弃了遮盖的设计，输入仍旧维持原来的句子，保证训练与微调过程的一致性。
- 在自注意力编码过程中将句子顺序打乱，从而使得模型能够学习上下文，充分利用信息。

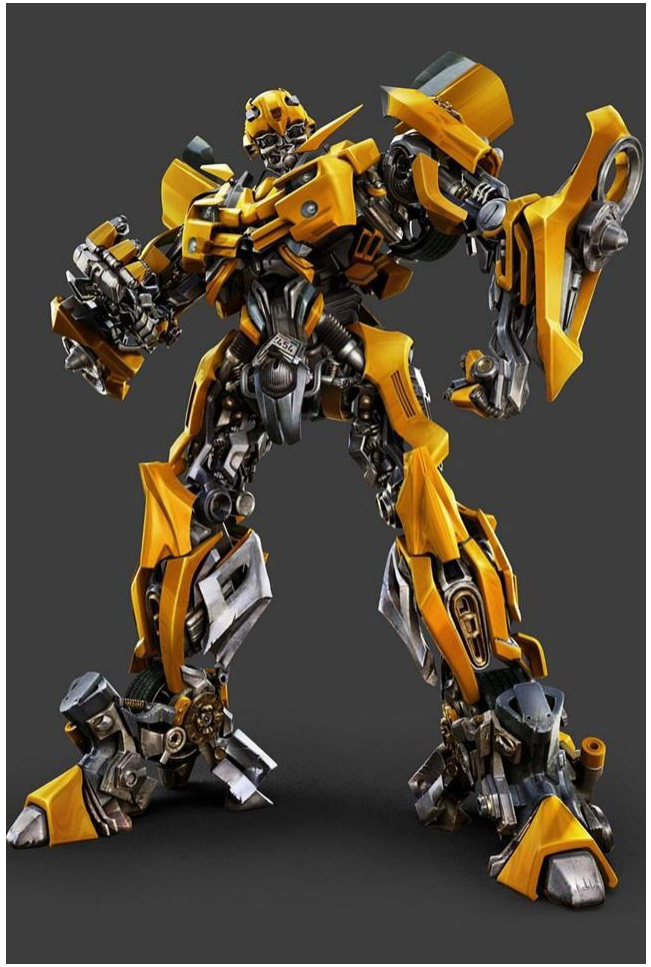
注意力机制在NLP上的发展历史

- 注意力->记忆力
- 人类智力的一大组成部分便是记忆力，是一种识别、保持、再认识和重现客观事物相应内容及经验的能力。
- 纵观近些年自然语言处理相关的深度学习研究，也是一路在机器的记忆力上做文章。

RNN	RNN+att	Transformer	Bert	XLNet
记忆	侧重的记忆	空间上的完全记忆	上下文的完全记忆	解决训练与微调的矛盾

Transformer详解

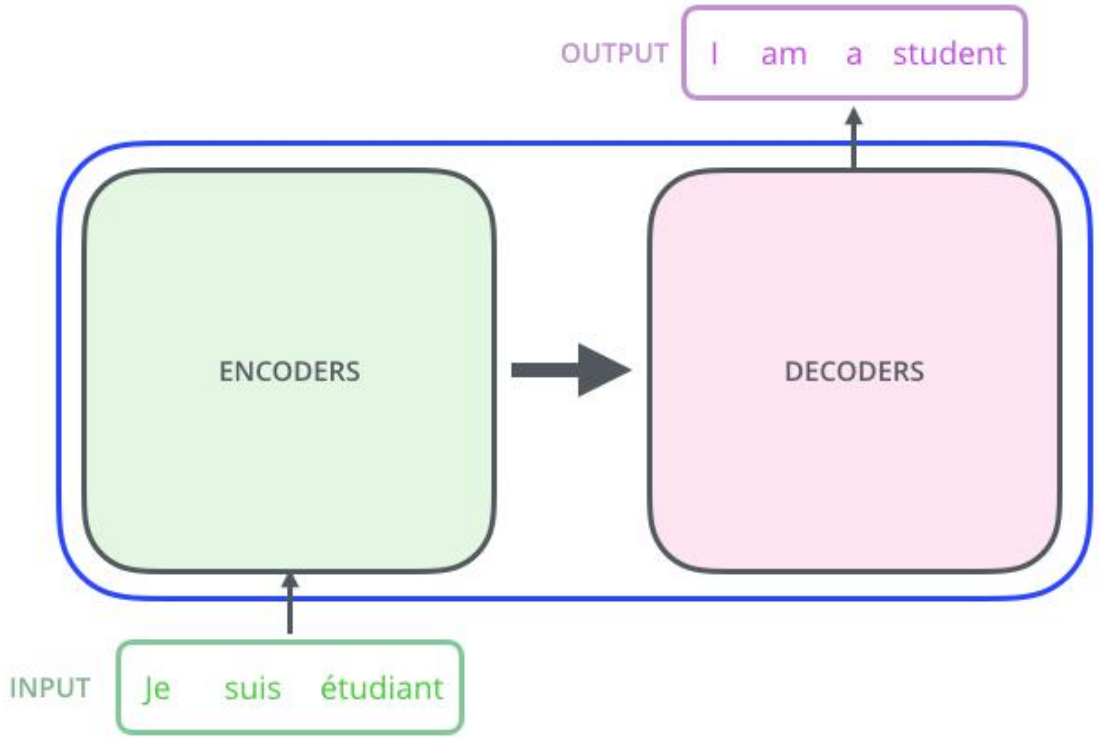




Transformer详解

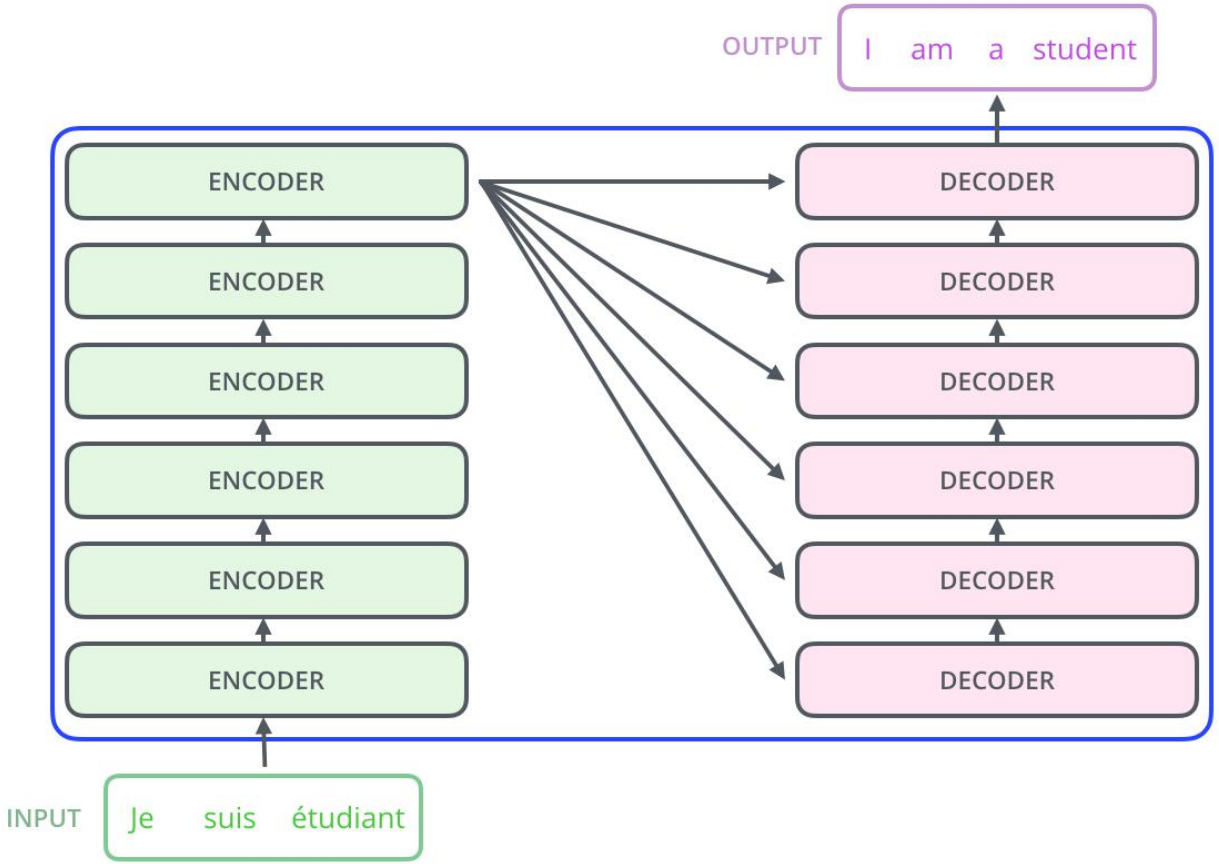
Tramsformer来源于哈佛大学所提出的论文Attention is All You Need，是一种区别于循环神经网络以及一般注意力机制的全新型模型，主要核心：自注意力机制。
机器翻译应用：输入为一种语言的句子，以另一种语言输出其翻译。

Transformer详解



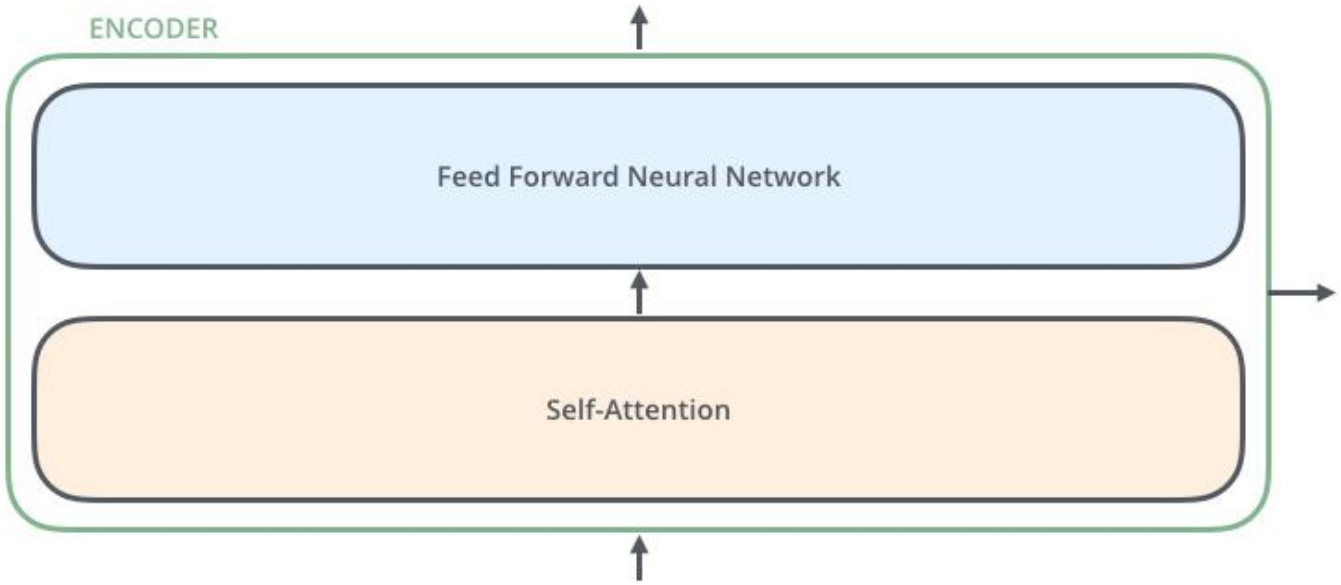
同样，在Tramsformer的内部也是由encoder和decoder构成。
encoder和decoder都是基于自注意力机制构成的结构（非RNN）。

Transformer详解



注：编码器和解码器都是一组编码组件和一组解码组件
原论文中所提出的具体数目是6个

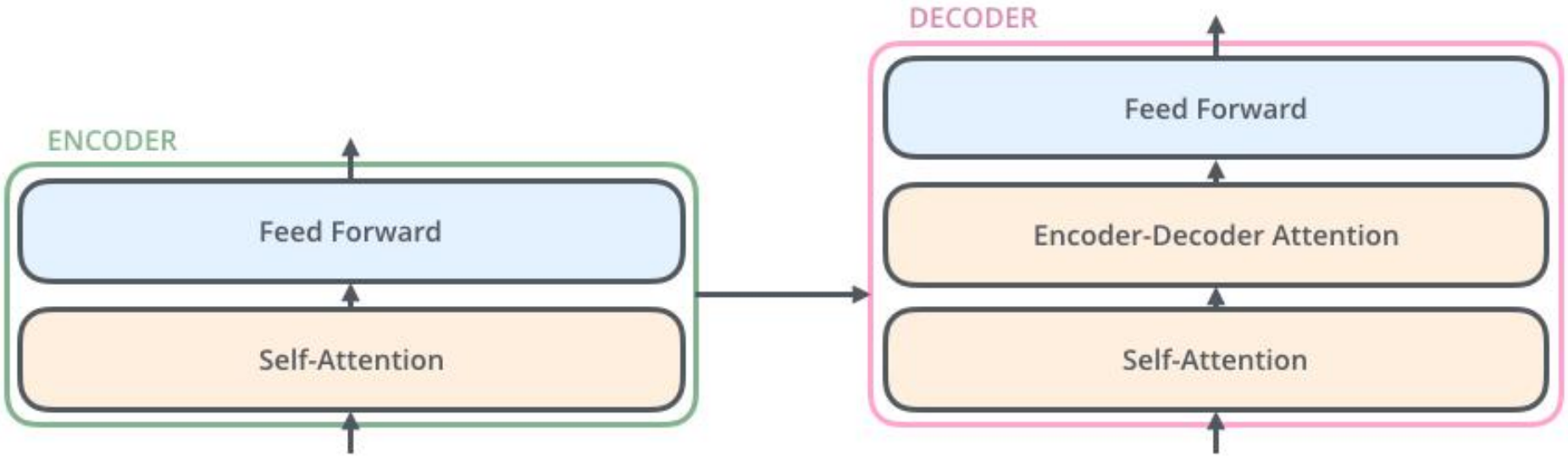
Transformer详解



对于每个编码器来说，他们的结构均相同（但它们不共享权重）。
每一层都分为两个子层：

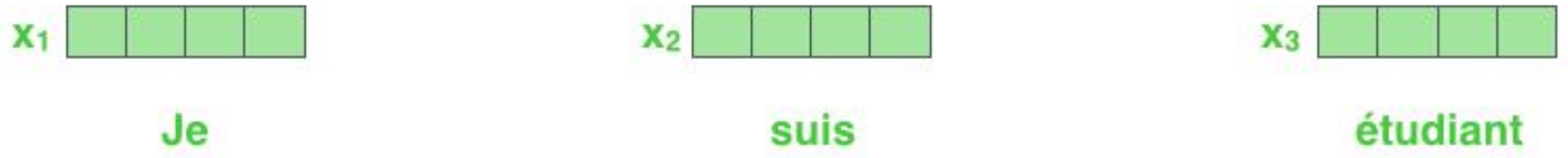
- 编码器的输入首先流经自注意力层，该层可以编码出同一个句子中不同词语之间的关系。
- 自注意力层的输出之后便是一个简单的全连接神经网络。

Transformer详解



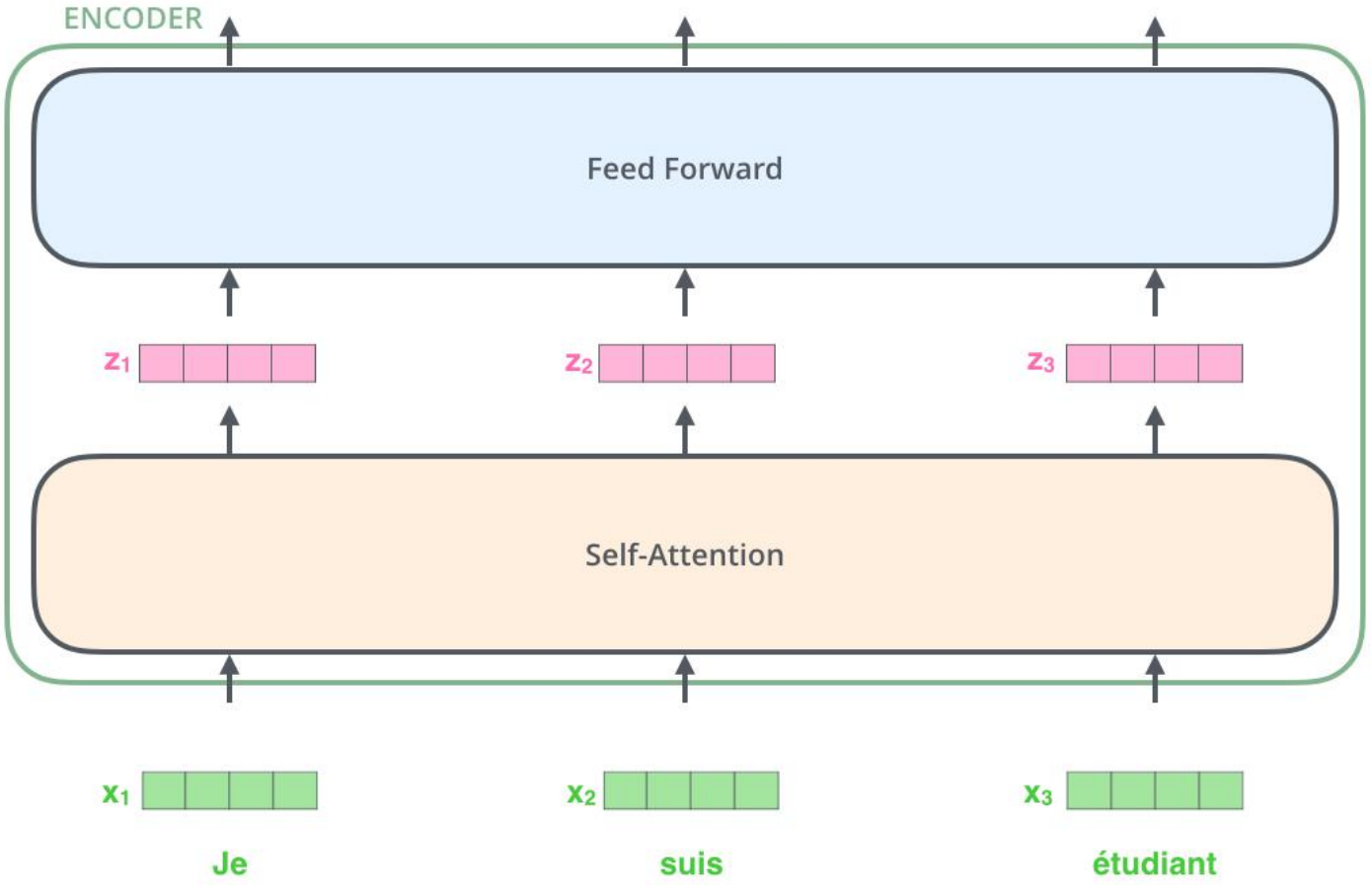
解码器：还有encoder和decoder之间的注意力层，可以帮助解码器将注意力集中在输入语句的相关部分上（类似于seq2seq模型中的注意力）。

Transformer详解



首先：将每个输入的词汇转化成向量，也称为词嵌入/词向量（word embedding），嵌入仅发生在最底部的编码器中。
输入数据大小：batch_size, word_embedding_size, seq_len
为了方便演示，这里假设词嵌入的长度为4。

Transformer详解



每个词嵌入都流经编码器的两层：

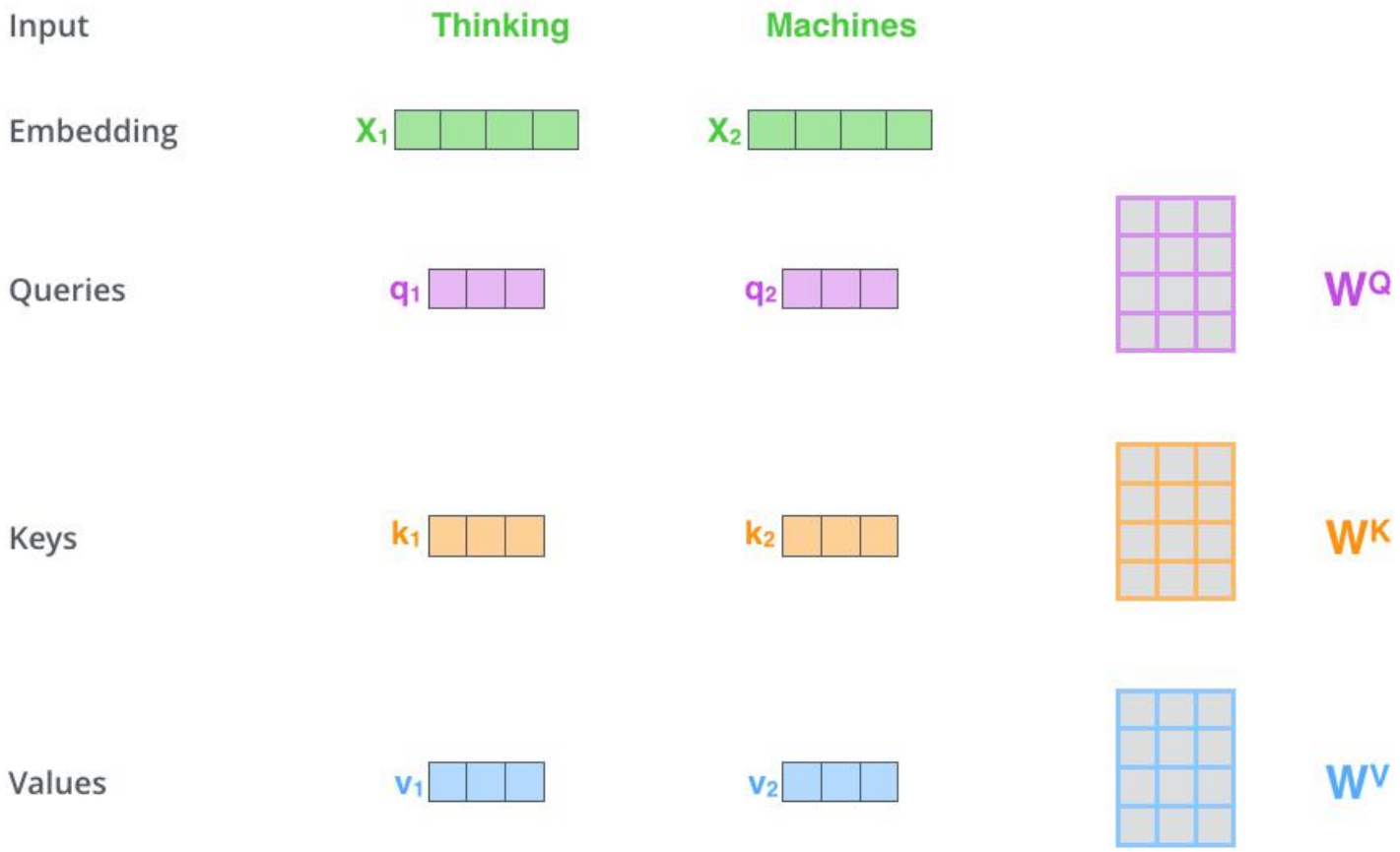
- “自注意力”层
- 前馈神经网络

然后将输出向上发送到下一个编码器。

Transformer的关键属性：每个位置的单词都流经编码器中自己的路径(计算无先后关系)。

所以，运算速度比一般的循环神经网络（RNN）要快很多。

Transformer详解

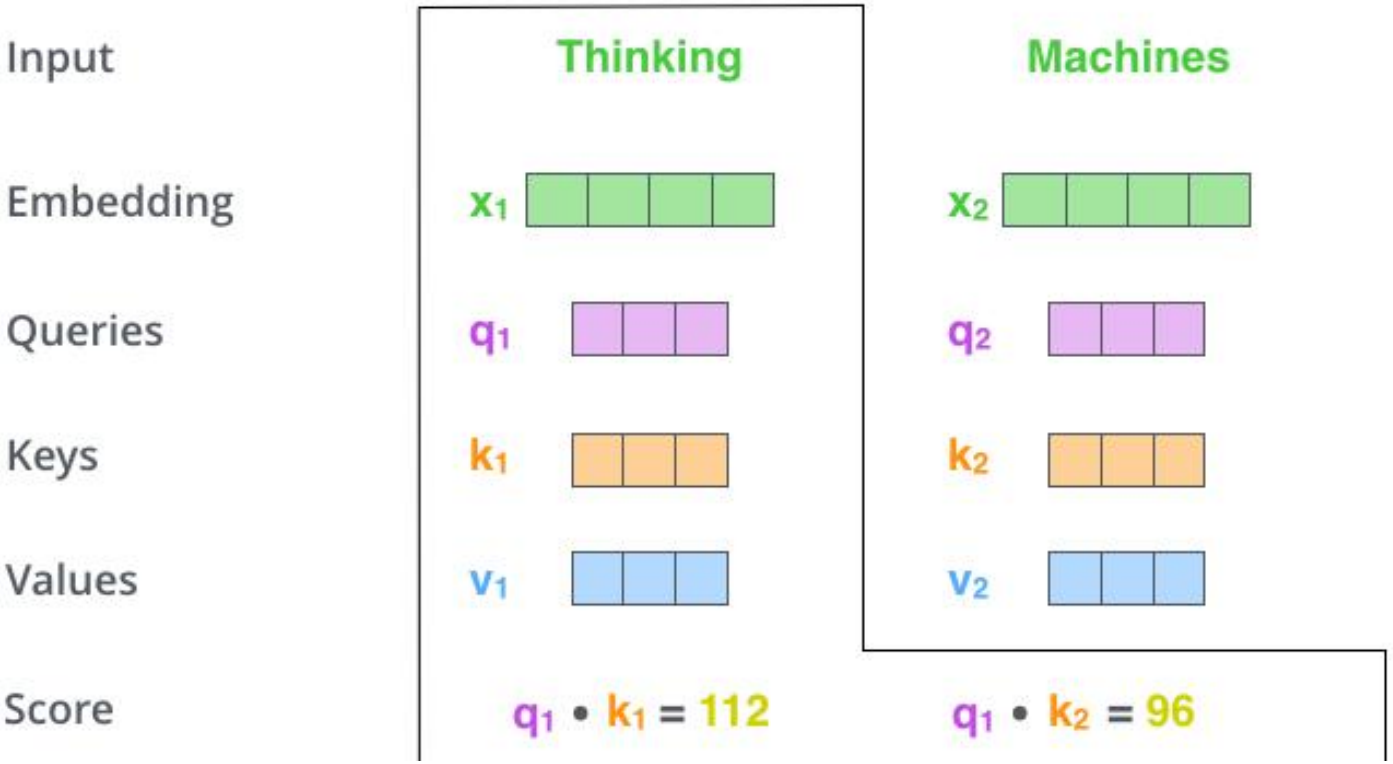


如何计算自注意力

对于每个单词，创建一个查询向量，一个键向量和一个值向量。称为qkv， 通过将嵌入乘以我们在训练过程中训练的三个参数矩阵，也就是三个W，来创建这些向量。

（注意：原文中，它们的维数为64，而词嵌入和编码器输入/输出矢量的维数为512。）

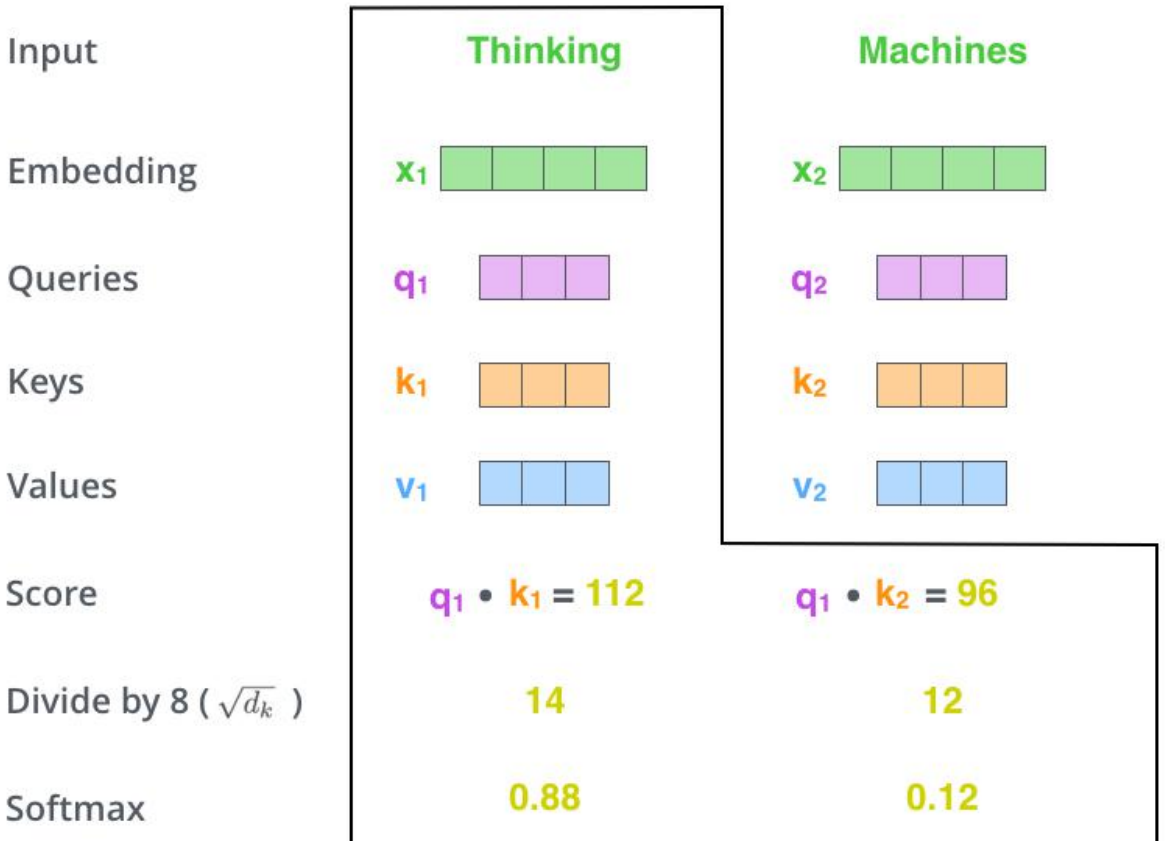
Transformer详解



如何计算自注意力

通过将q向量与各个单词的k向量的乘积来计算分数。
分数越高，说明关系越密切。

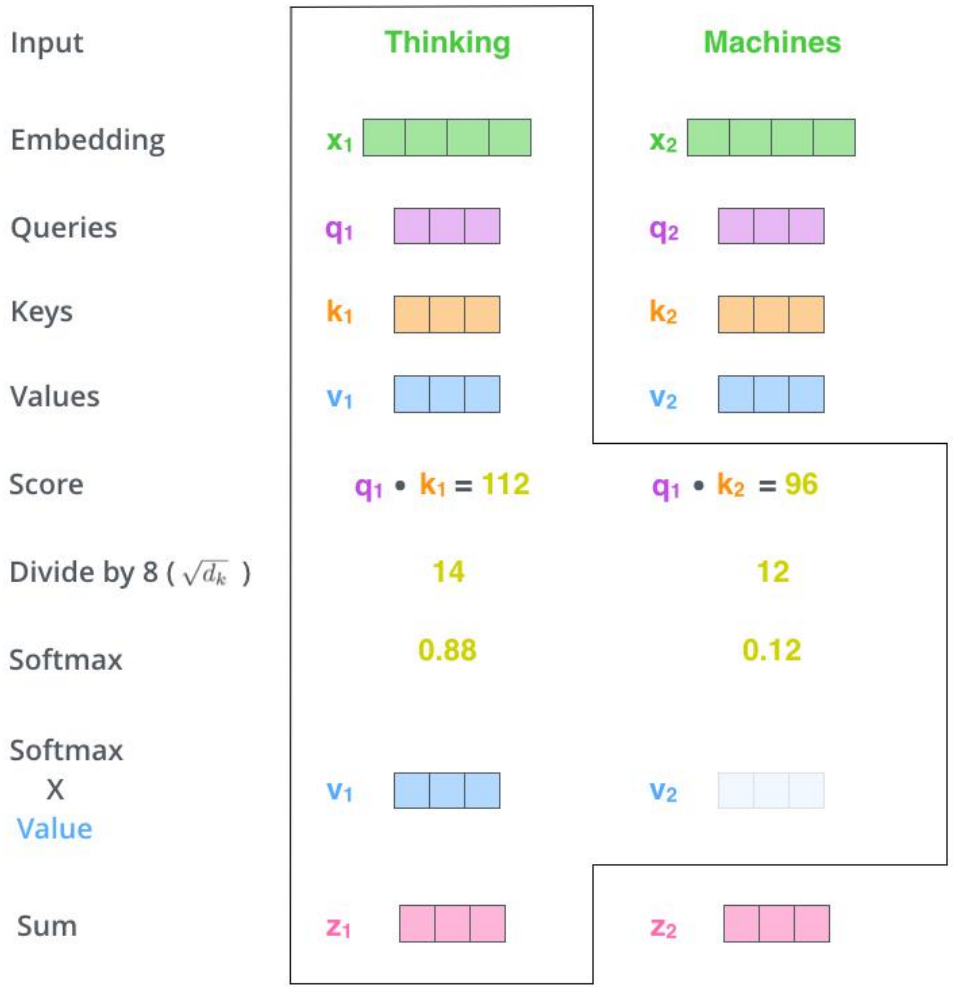
Transformer详解



如何计算自注意力

将分数缩小，然后通过softmax操作转化为概率。

Transformer详解

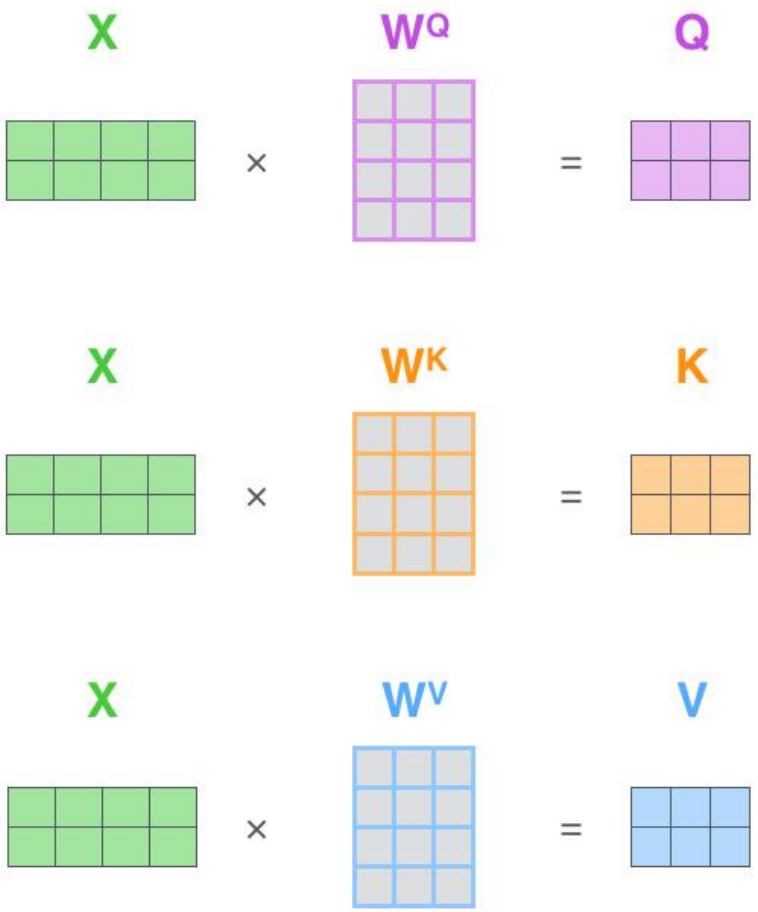


如何计算自注意力

将每个v向量乘以概率值并求和。
生成的向量是发送到前馈神经网络。

（注：在实际实现中，此计算以矩阵形式进行，以加快处理速度。）

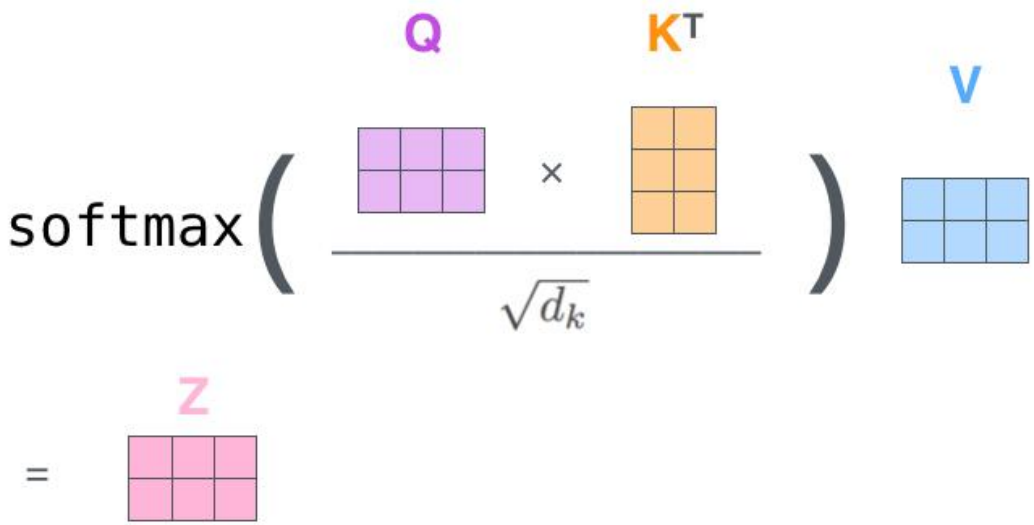
Transformer详解



自注意力的矩阵计算

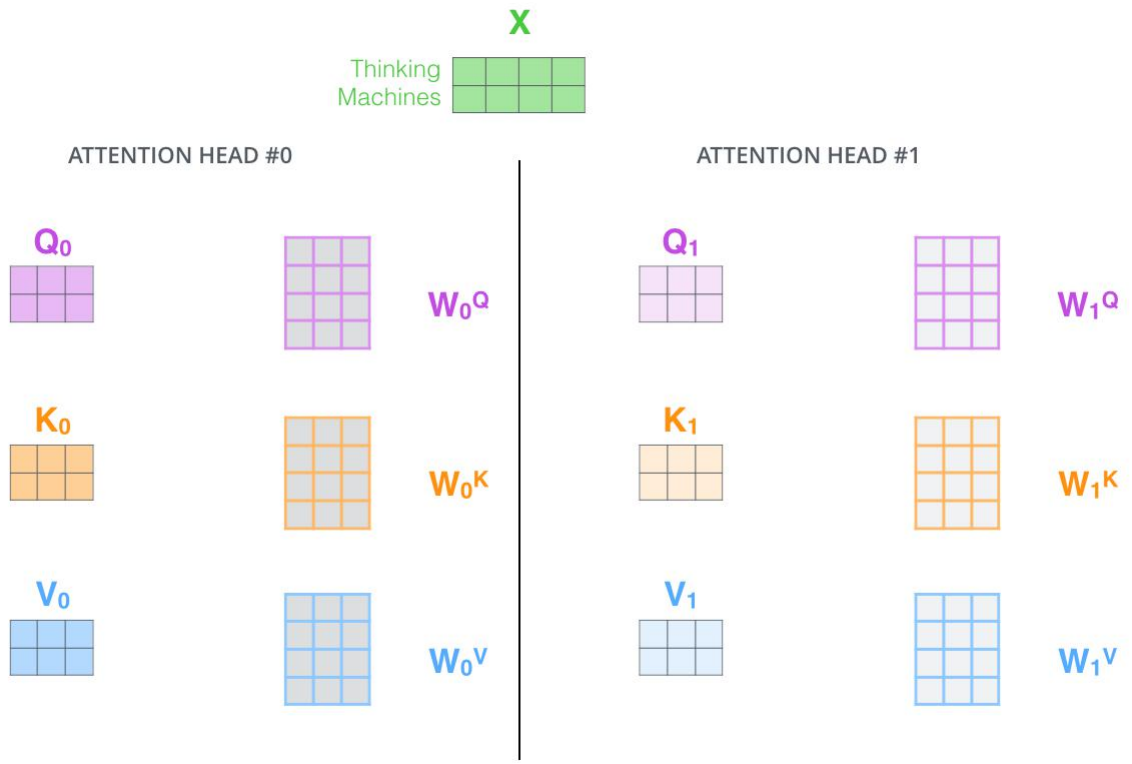
计算查询，键和值矩阵。
将所有词嵌入整合到到矩阵X中，然后将其乘以权重矩阵（WQ，WK，WV）。

Transformer详解



自注意力的矩阵计算图示

Transformer详解

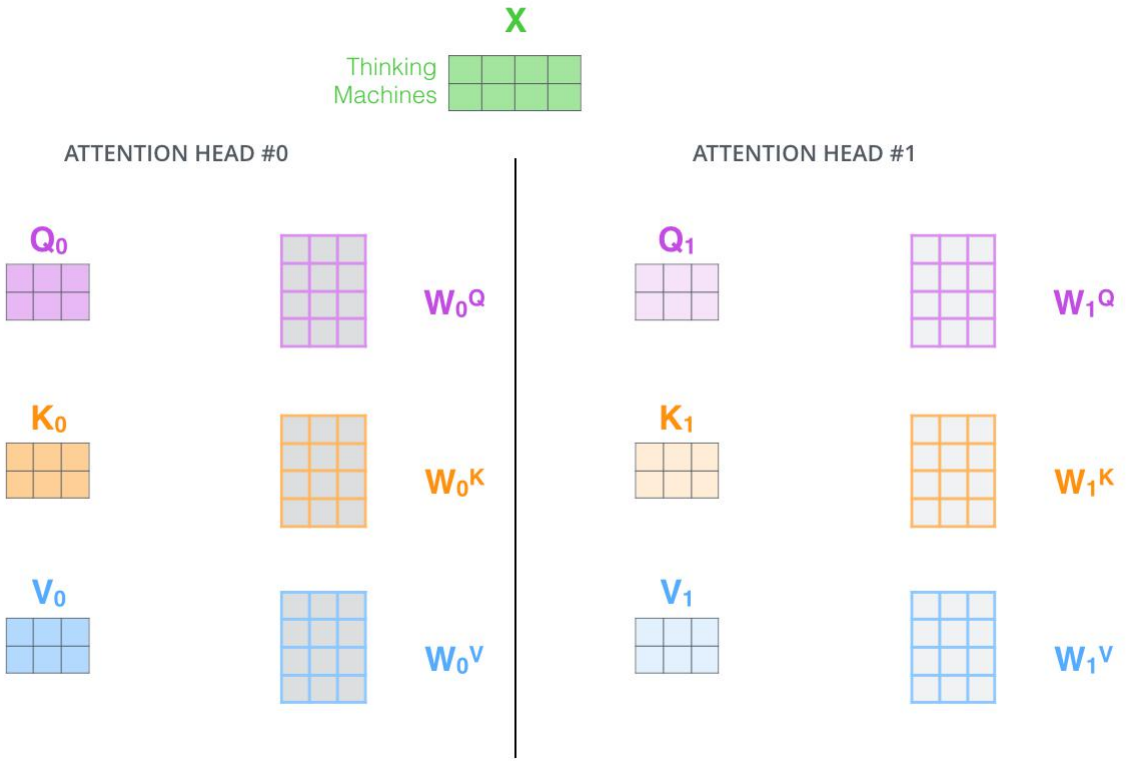


多头机制

论文通过添加一种称为“多头”注意力的机制，进一步完善了自注意力层。

扩展了模型专注于不同层面的能力。

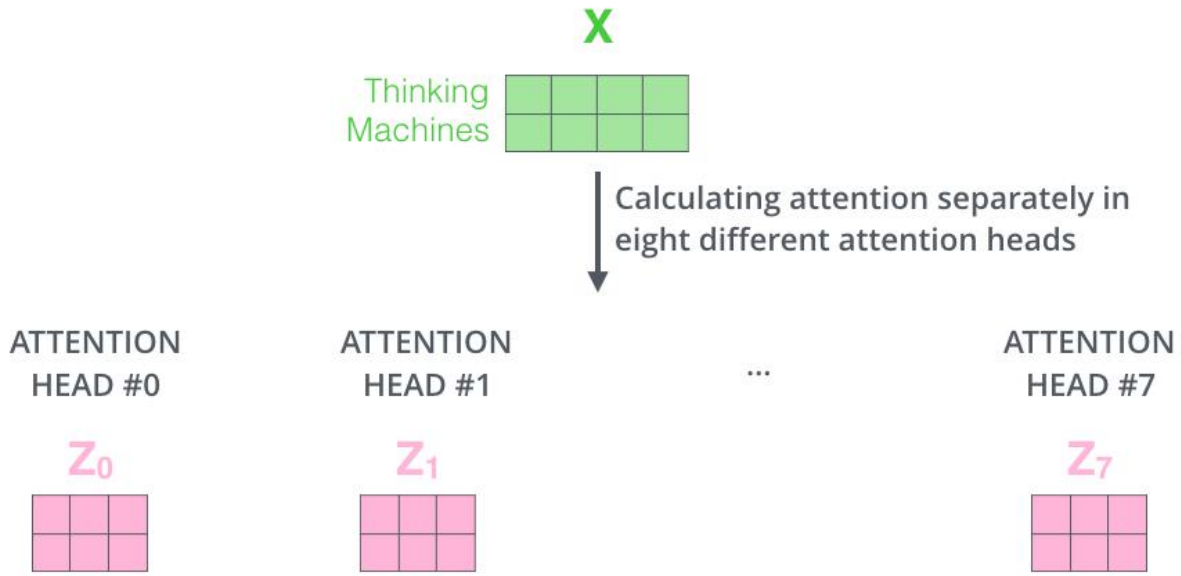
Transformer详解



多头机制

提供了多个“表示子空间”。
在多头机制下，有多组查询/键/值权重矩阵（Transformer使用八个头，因此每个编码器/解码器最终得到八组）。

Transformer详解



多头机制

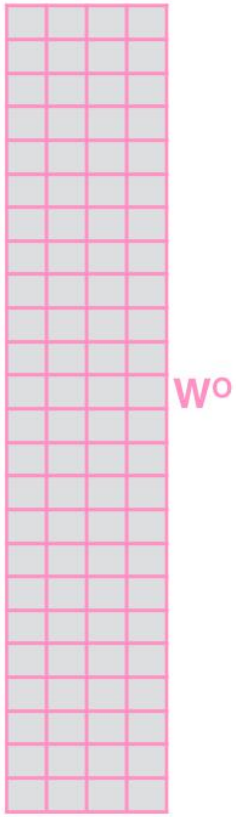
执行相同的自注意力计算，最终将得到八个不同的Z矩阵。

Transformer详解

1) Concatenate all the attention heads

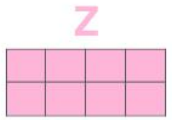
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

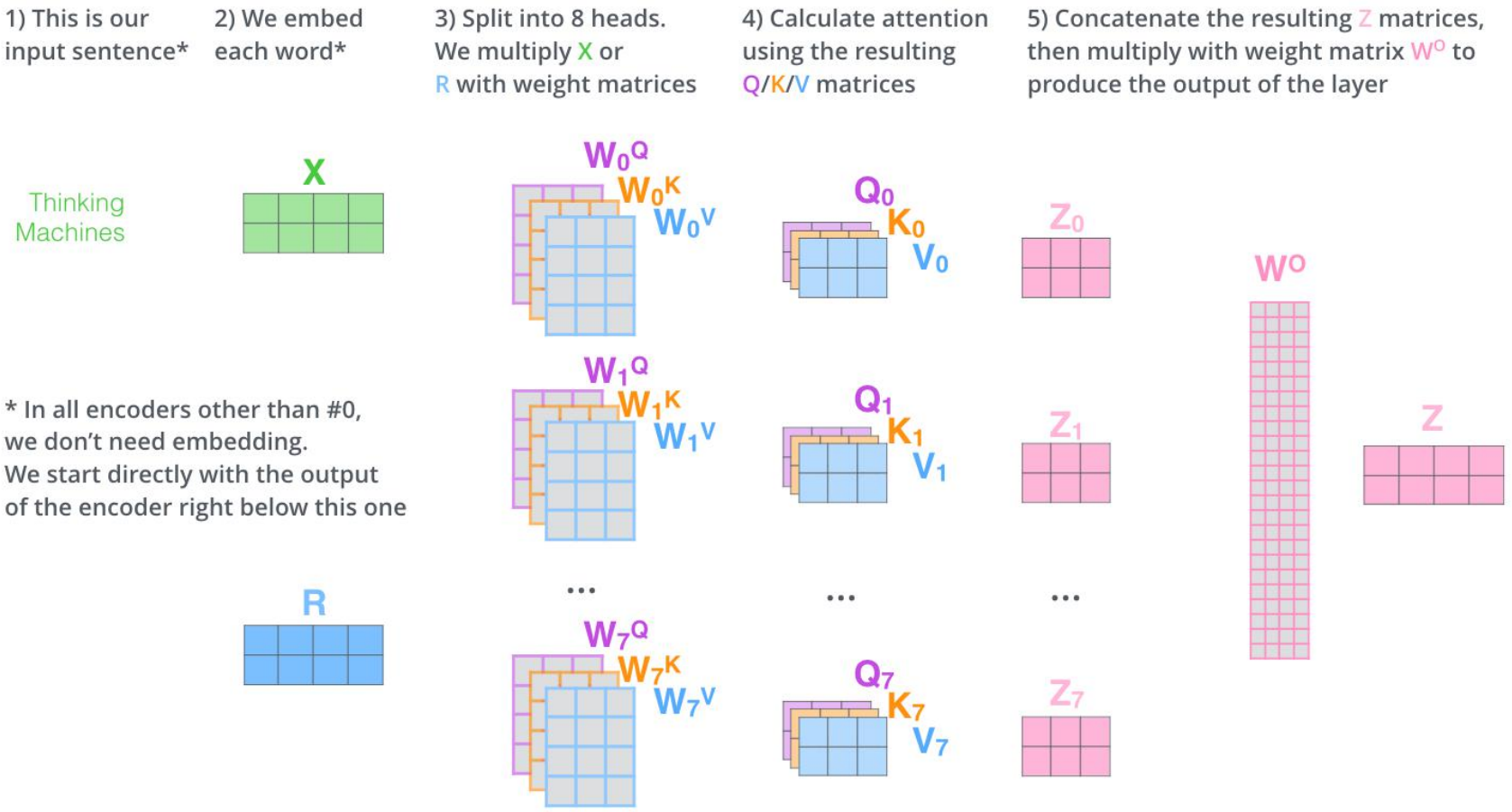
=



多头机制

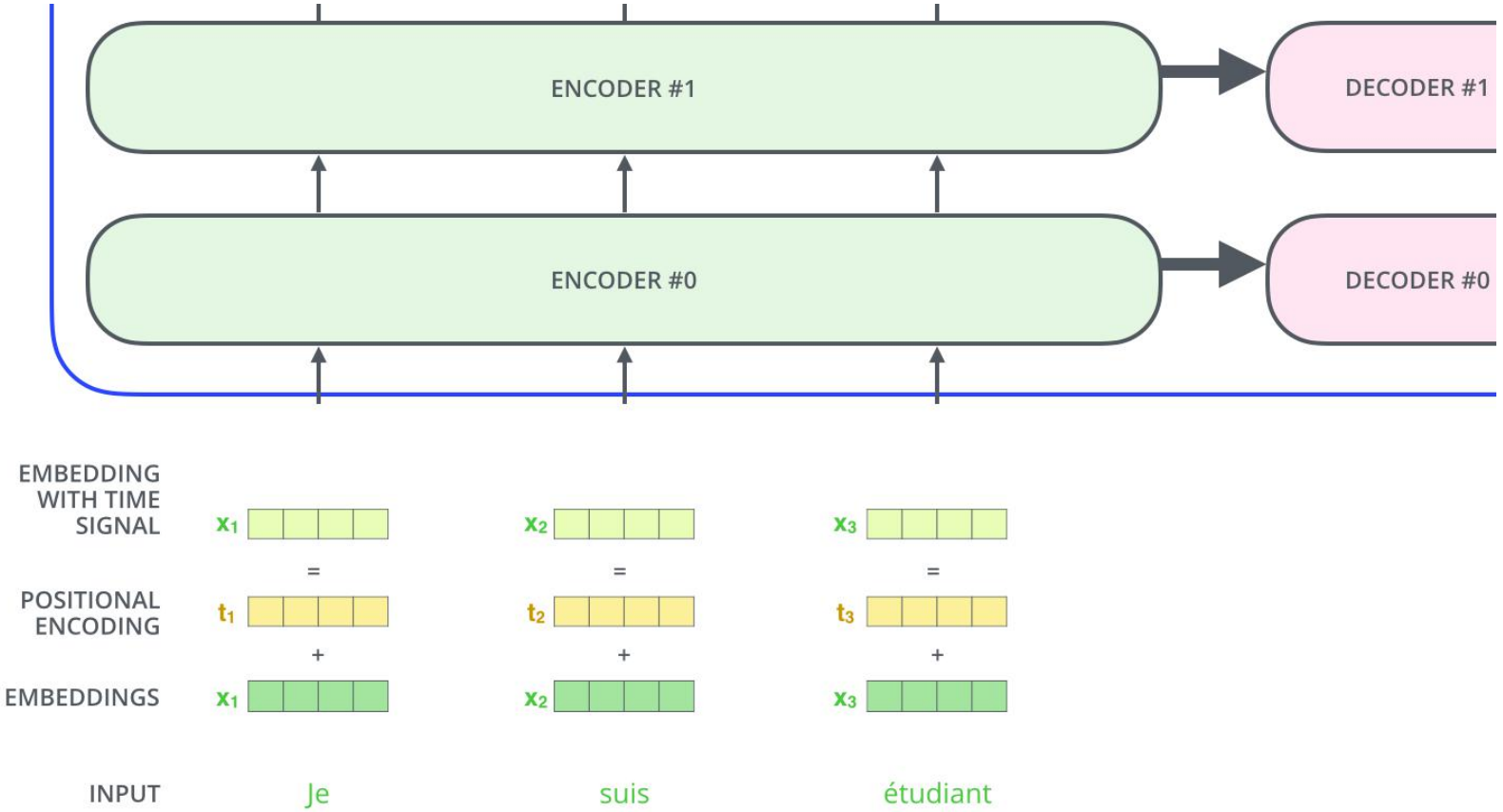
合并矩阵，然后乘以权重矩阵，转化为一个Z。

Transformer详解


1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer
Thinking
Machines
X
R
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one
W0Q W0K W0V
W1Q W1K W1V
W7Q W7K W7V
Q0 K0 V0
Q1 K1 V1
Q7 K7 V7
Z0
Z1
Z7
WO
Z

自注意力全景图

Transformer详解



使用位置编码表示序列的顺序

各单词并行运算，没有考虑到顺序关系，如何引入顺序特征？

答：对每个单词进行顺序编码。

Transformer详解

使用位置编码表示序列的顺序（sin,cos有相对位置性）

pos表示单词在句子中的位置
2i表示偶数维度
2i+1表示奇数维度

输入序列位置编码$Pos_Enc\,(inputs_position) \in \mathbb{R}^{N \times d_{model}}$，
其中，$inputs_position = (1, 2, \cdots, p, \cdots, N)$为输入序列中输入符号对应的位置序号；

$$Pos\_Enc_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right)$$

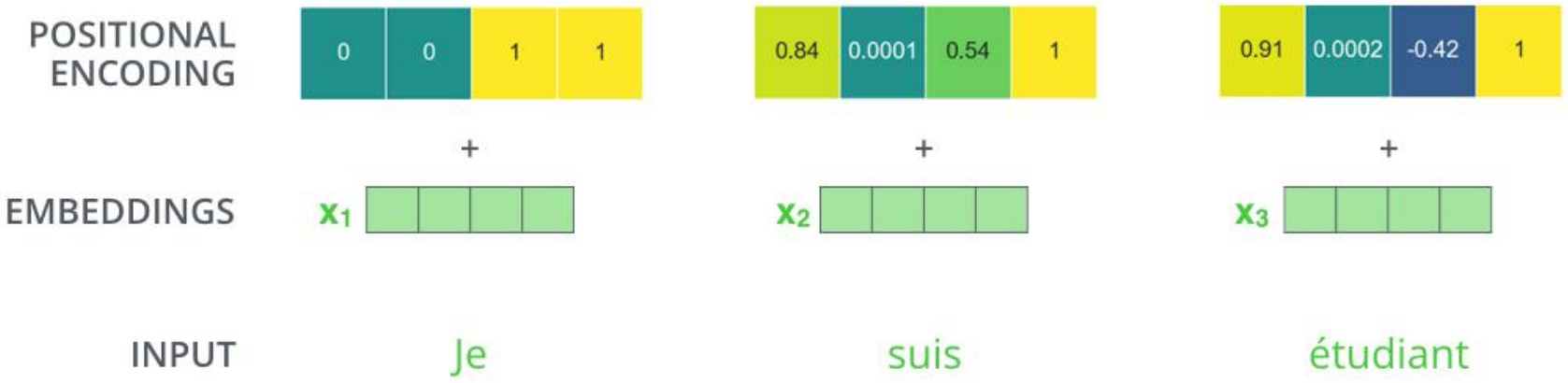
$$Pos\_Enc_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right)$$

其中，$pos \in inputs\_position, i \in (0, 1, \cdots, d_{model}/2)$。

Transformer详解

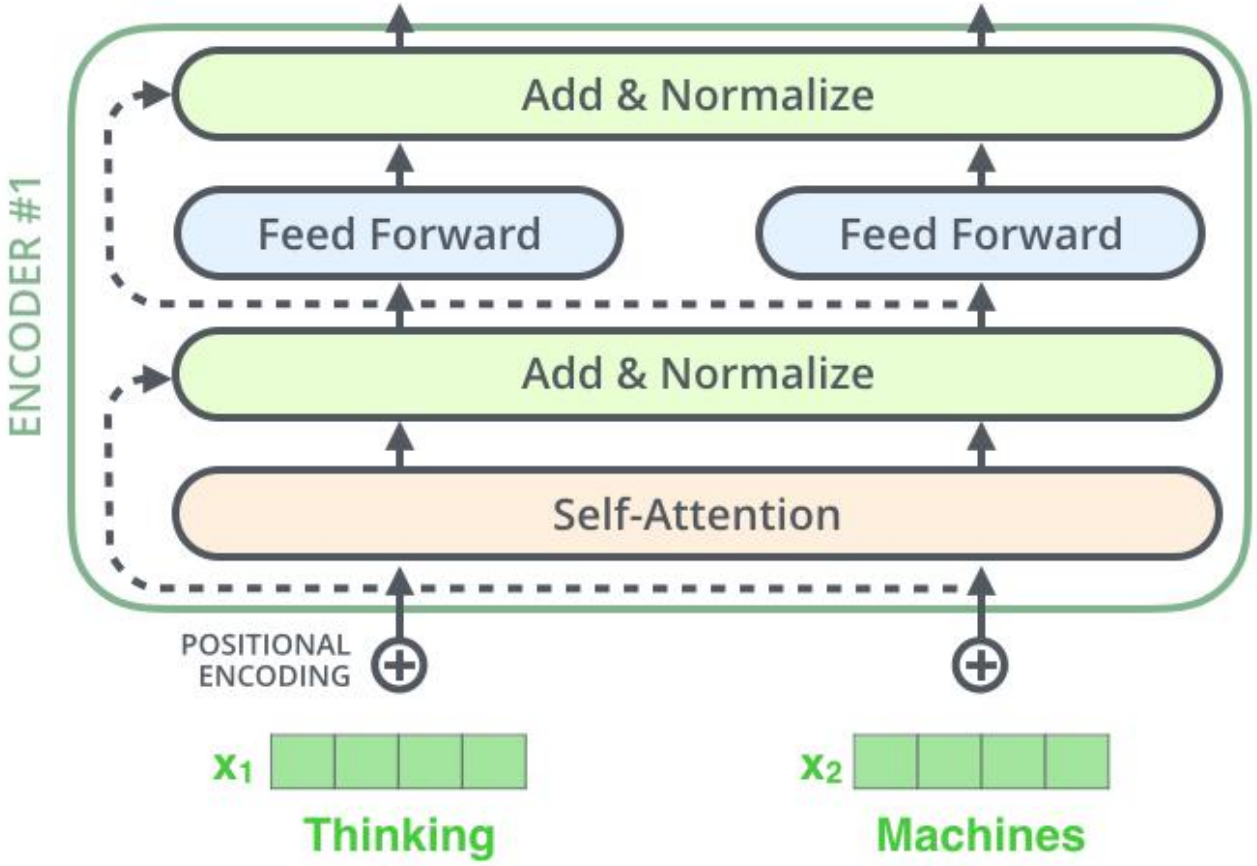
使用位置编码表示序列的顺序

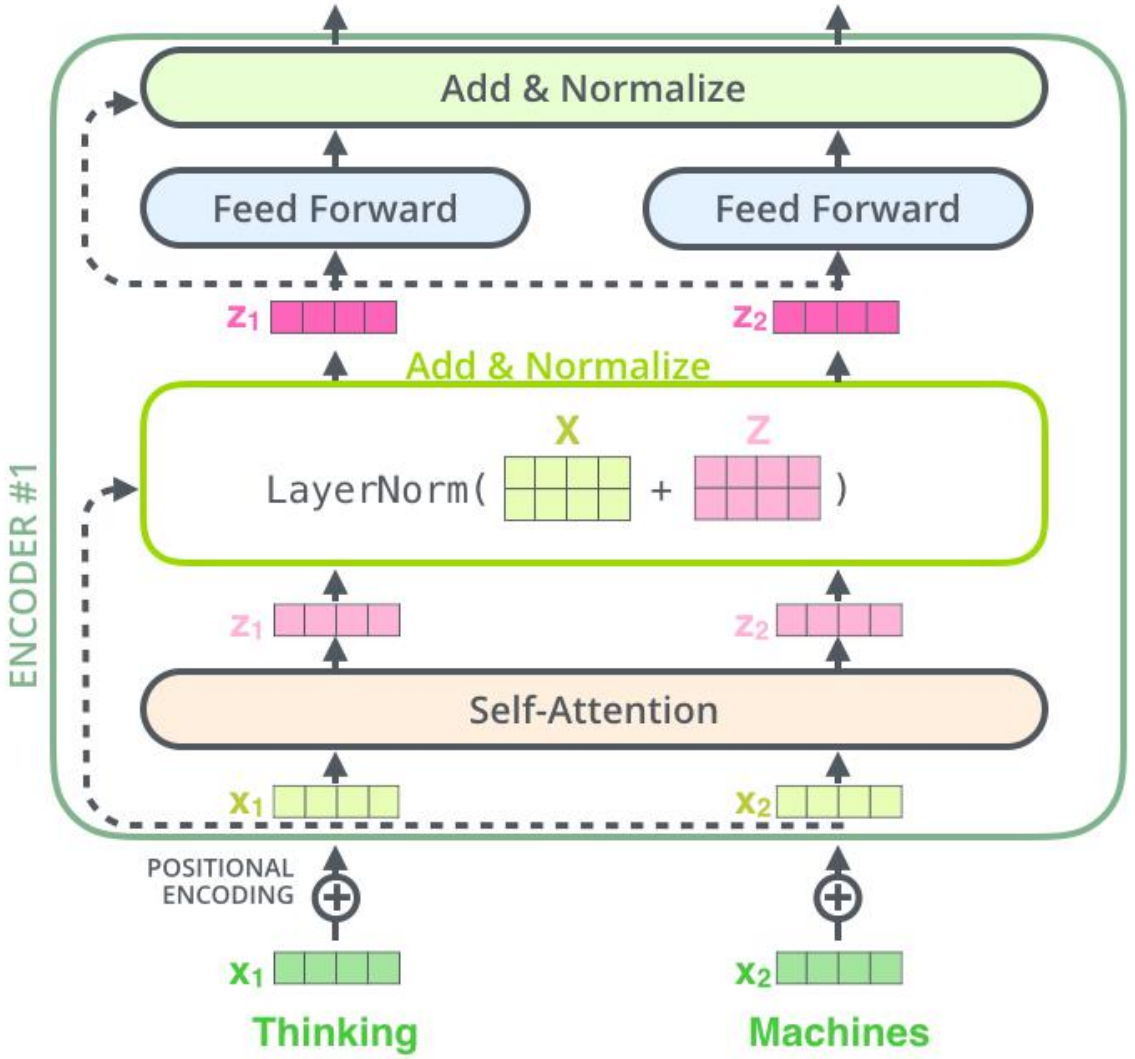
假设嵌入的维数为4，则实际的位置编码应如下所示：



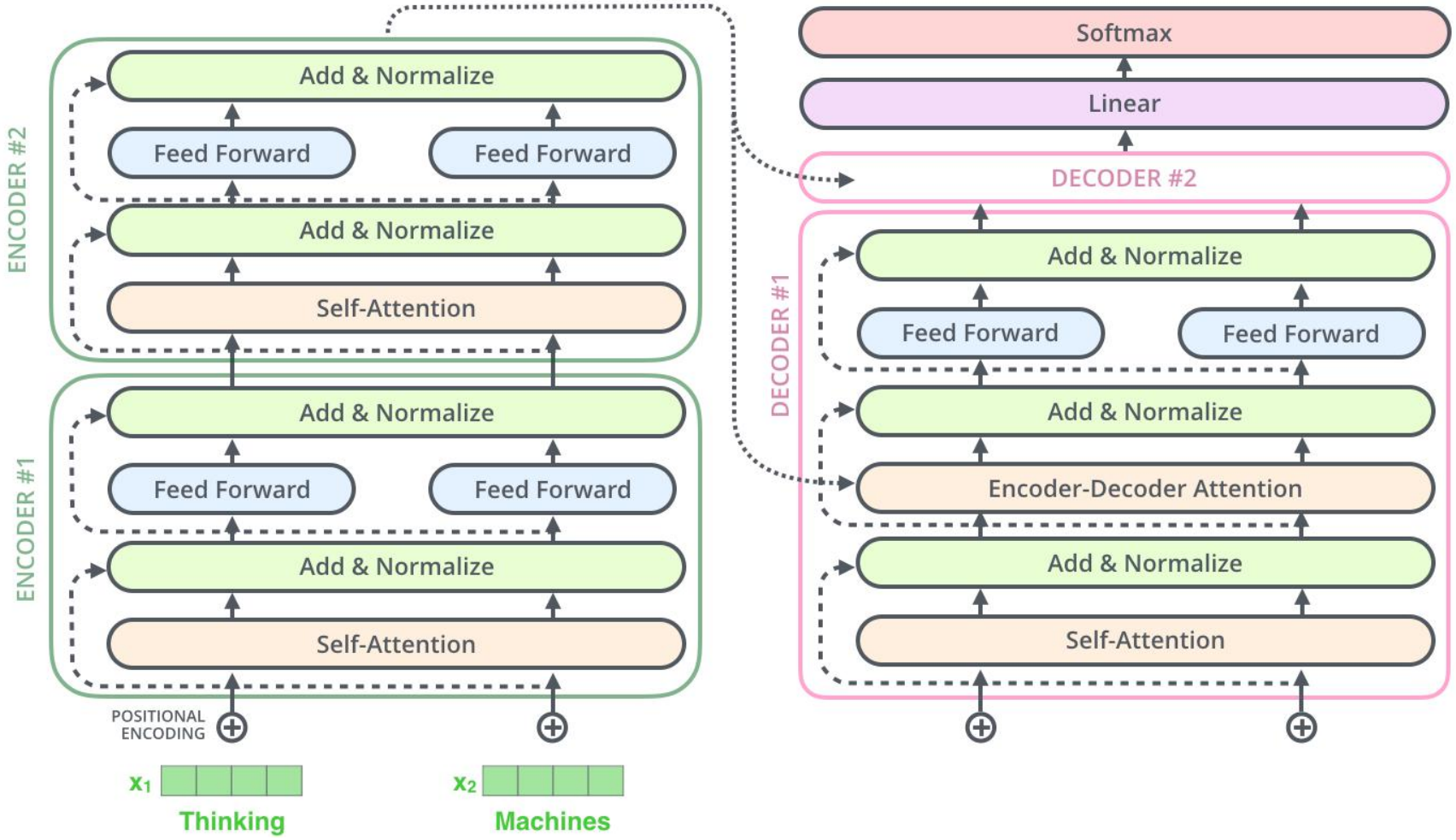
Transformer详解

几个细节：残差，归一化



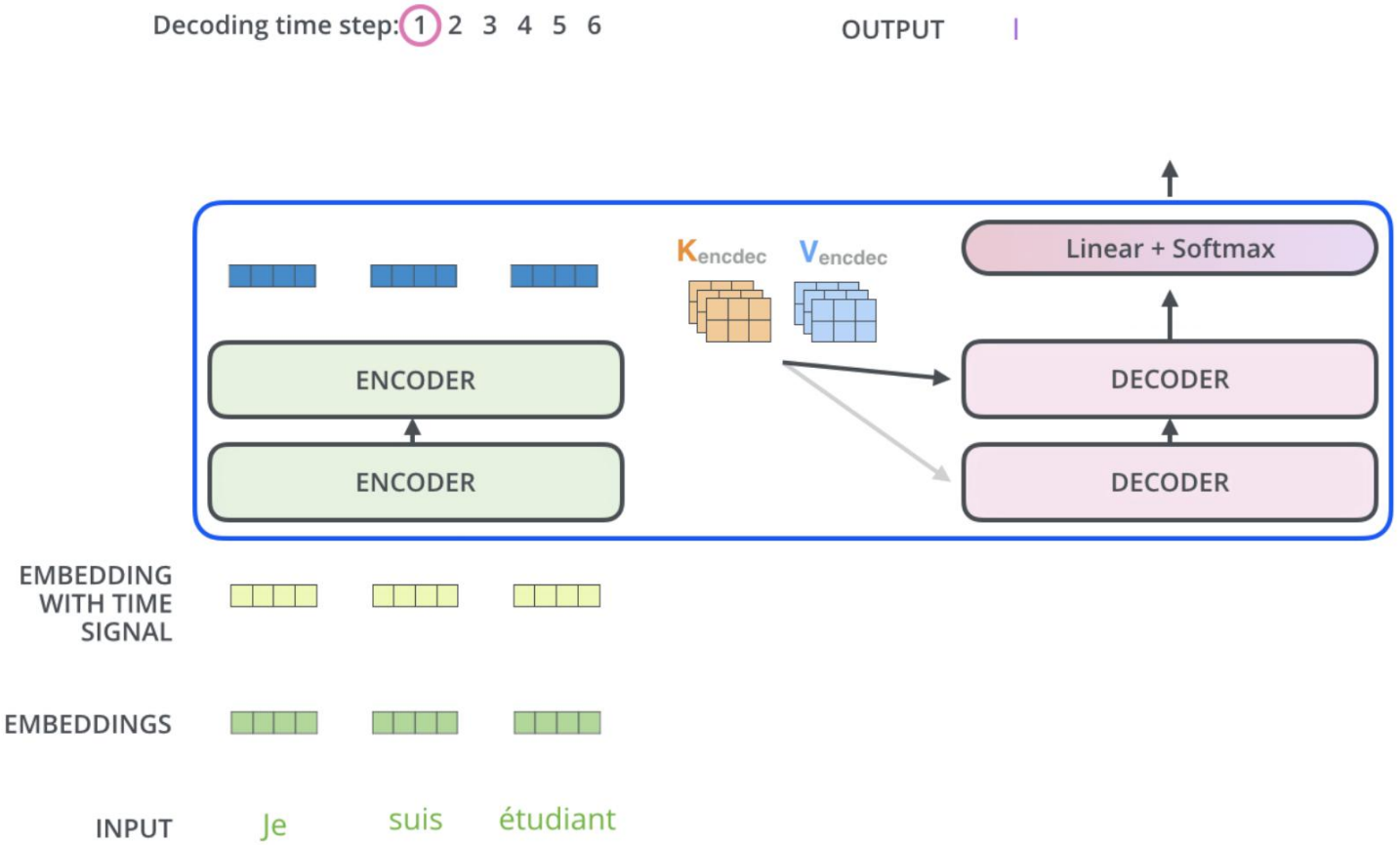


Transformer详解



2个堆叠式编码器和解码器组成的Transformer

Transformer详解

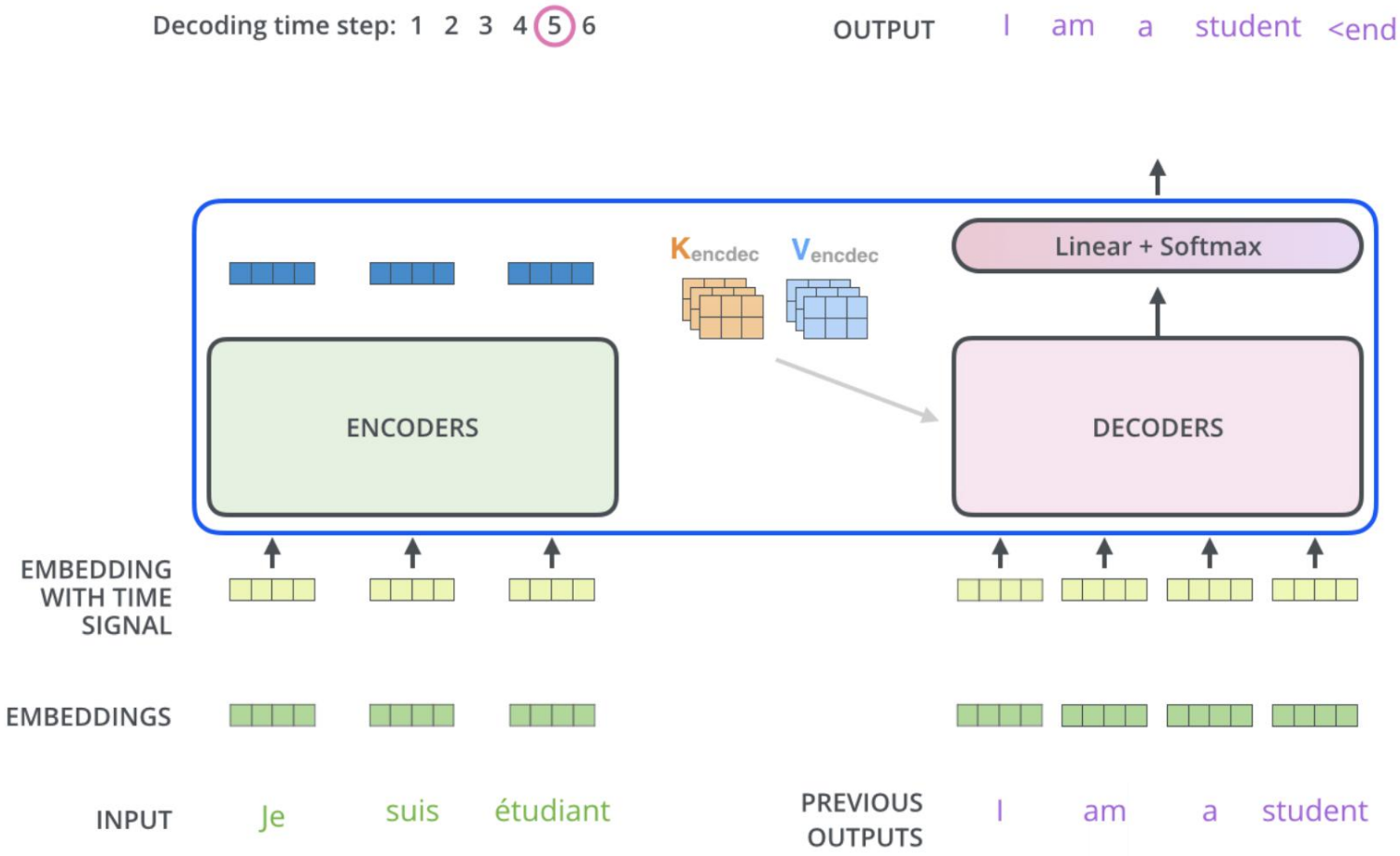


编码器，解码器如何协同工作。

- 编码器首先处理输入序列。
- 顶部编码器的输出转换为注意力向量K和V的集合。
- 每个解码器将在其“编码器-解码器注意”层中使用它们，这有助于解码器将注意力集中在输入序列中的适当位置。

（解码器结构：自注意力+编码器-解码器注意+前馈层）

Transformer详解



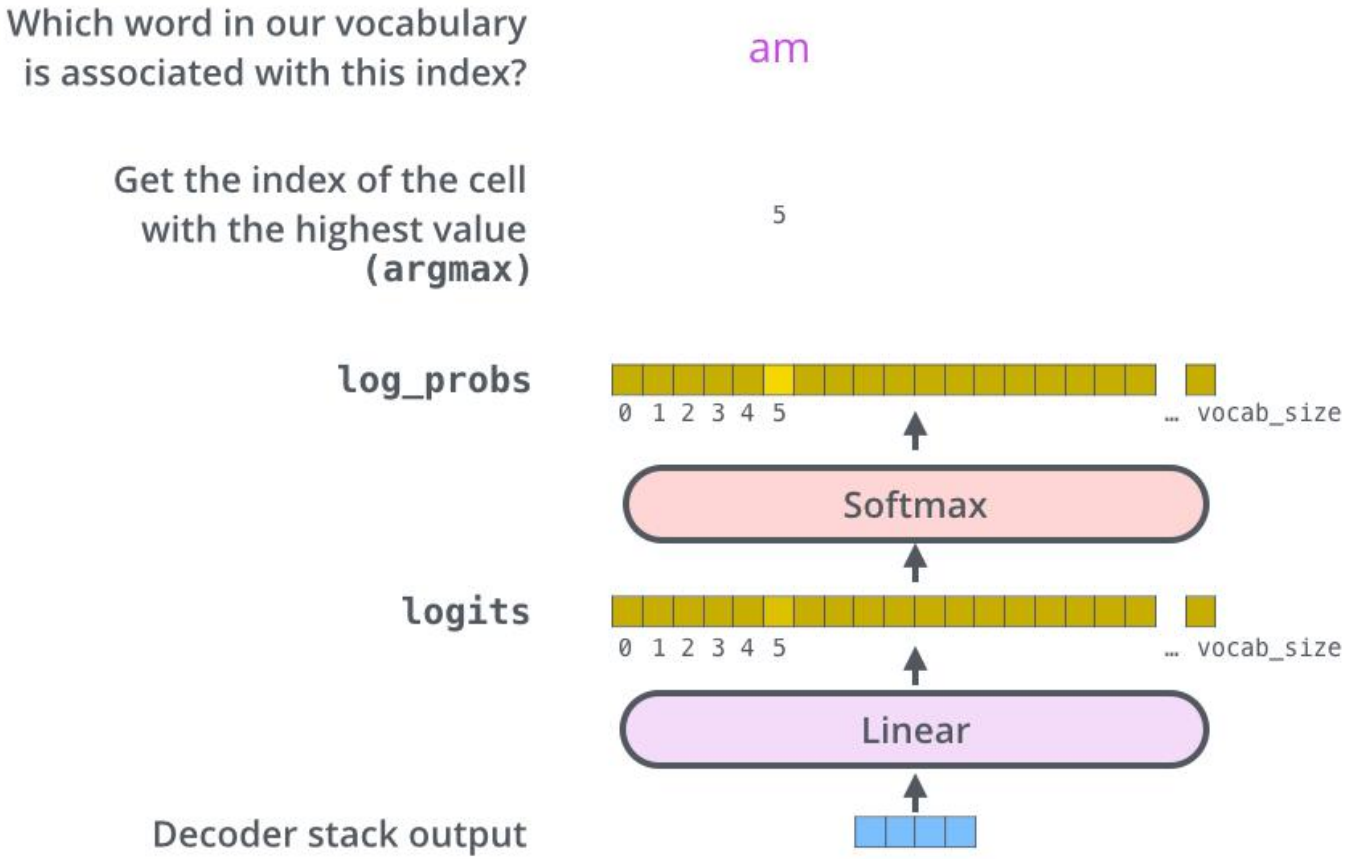
解码器端

解码器中的自注意力层与编码器中的自注意力层的不同：

掩码：在解码器中，仅允许自注意力层参与输出序列中的较早位置。

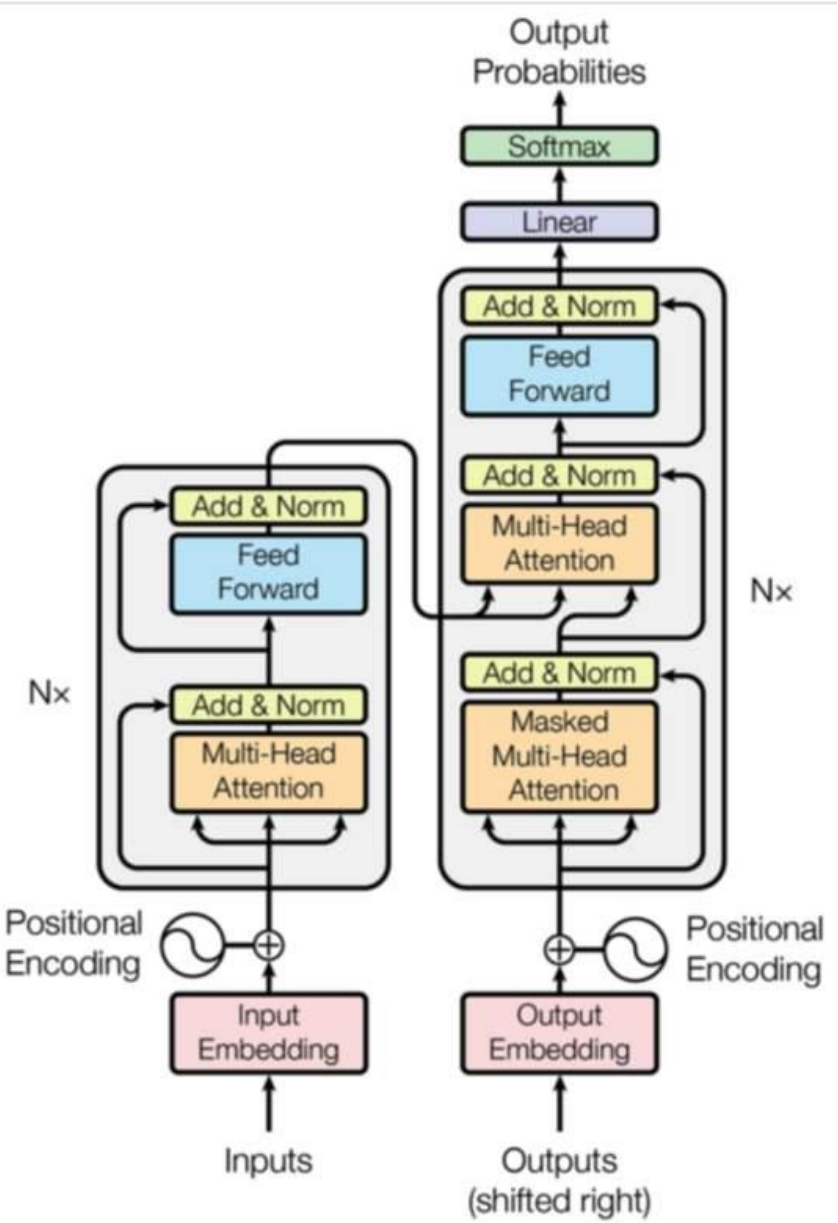
“编码器-解码器注意”层的工作方式与多头自注意力类似，不同之处在于它从编码器的输出中获取键和值矩阵（context）。

Transformer详解



最终的线性层和Softmax层(词典大小的多分类)。

Transformer总结



Transformer总结

输入序列、目标序列与输出序列

输入序列$inputs = \left(i_1, i_2, \cdots, i_p, \cdots, i_N\right)$，其中$i_p \in \mathbb{N}^*$为输入符号表中的序号。

目标序列$targets = \left(t_1, t_2, \cdots, t_q, \cdots, t_M\right)$，其中$t_q \in \mathbb{N}^*$为目标符号表中的序号。

$$outputs\_probabilities = Transformer\left(inputs, targets\right)$$

其中，$outputs_probabilities = \left(o_1, o_2, \cdots, o_q, \cdots, o_M\right)$为预测序列，$o_q \in \mathbb{N}^*$为目标符号表中的序号。

Transformer总结

词嵌入与位置编码

输入序列词嵌入与位置编码

输入序列词嵌入$Embedding\left(inputs\right) \in \mathbb{R}^{N\times d_{model}}$，其中，$N$为输入序列长度，$d_{model}$为词嵌入维度。

输入序列位置编码$Pos_Enc\left(inputs_position\right) \in \mathbb{R}^{N\times d_{model}}$，
其中，$inputs_position = (1, 2, \cdots, p, \cdots, N)$为输入序列中输入符号对应的位置序号；

$$Pos\_Enc_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right)$$

$$Pos\_Enc_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right)$$

其中，$pos \in inputs\_position, i \in (0, 1, \cdots, d_{model}/2)$。

目标序列词嵌入与位置编码

目标序列词嵌入$Embedding\left(targets\right) \in \mathbb{R}^{M\times d_{model}}$，其中$M$为目标序列长度，$d_{model}$为词嵌入维度。

目标序列位置编码$Pos_Enc\left(targets_position\right) \in \mathbb{R}^{M\times d_{model}}$，
其中，$targets_position = (1, 2, \cdots, q, \cdots, M)$为目标序列的位置序号。

Transformer总结

编码器Encoder

编码器结构

编码器结构：

$$e_0 = Embedding(inputs) + Pos\_Enc(inputs\_position)$$
$$e_l = EncoderLayer(e_{l-1}), l \in [1, n]$$

其中，$e_0 \in \mathbb{R}^{N \times d_{model}}$ 为编码器输入，$EncoderLayer(\cdot)$为编码器层，n为层数，$e_l \in \mathbb{R}^{N \times d_{model}}$ 为第l层编码器层输出。

编码器层EncoderLayer：

$$e_{mid} = LayerNorm(e_{in} + MultiHeadAttention(e_{in}))$$
$$e_{out} = LayerNorm(e_{mid} + FFN(e_{mid}))$$

其中，$e_{in} \in \mathbb{R}^{N \times d_{model}}$ 为编码器层输入，$e_{out} \in \mathbb{R}^{N \times d_{model}}$ 为编码器层输出，$MultiHeadAttention(\cdot)$为多头注意力机制，$FFN(\cdot)$为前馈神经网络，$LayerNorm(\cdot)$为层归一化。

Transformer总结

输入向量序列$e_{in} = (e_{in1}, e_{in2}, \cdots, e_{inN}) \in \mathbb{R}^{N \times d_{model}}$，分别得到查询向量序列$Q = e_{in}$，键向量序列$K = e_{in}$，值向量序列$V = e_{in}$。

多头注意力机制

$$MultiHeadAttention\,(e_{in}) = MultiHead\,(Q, K, V) = Concat\,(head_1, \cdots, head_h)\,W^O$$

其中，多头输出$head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right)$，可学习的参数矩阵$W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v}, W^O \in \mathbb{R}^{hd_v \times d_{model}}$

使用缩放点积作为打分函数的自注意力机制

$$Attention\left(QW_i^Q, KW_i^K, VW_i^K\right) = softmax\left(\frac{QW_i^Q\left(KW_i^K\right)^\top}{\sqrt{d_k}}\right)VW_i^V$$

Transformer总结

解码器结构

$$d_0 = Embedding(targets) + Pos\_Enc(targets\_position)$$
$$d_l = DecoderLayer(d_{l-1}), l \in [1, n]$$
$$outputs\_probabilities = softmax(d_n W)$$

其中，$d_0 \in \mathbb{R}^{M \times d_{model}}$为解码器输入，$DecoderLayer(\cdot)$为解码器层，$n$为层数，$d_l \in \mathbb{R}^{M \times d_{model}}$为第$l$层解码器层输出，$W \in \mathbb{R}^{M \times tgt_vocab_size}$输入输出参数矩阵，$softmax(\cdot)$为softmax层。

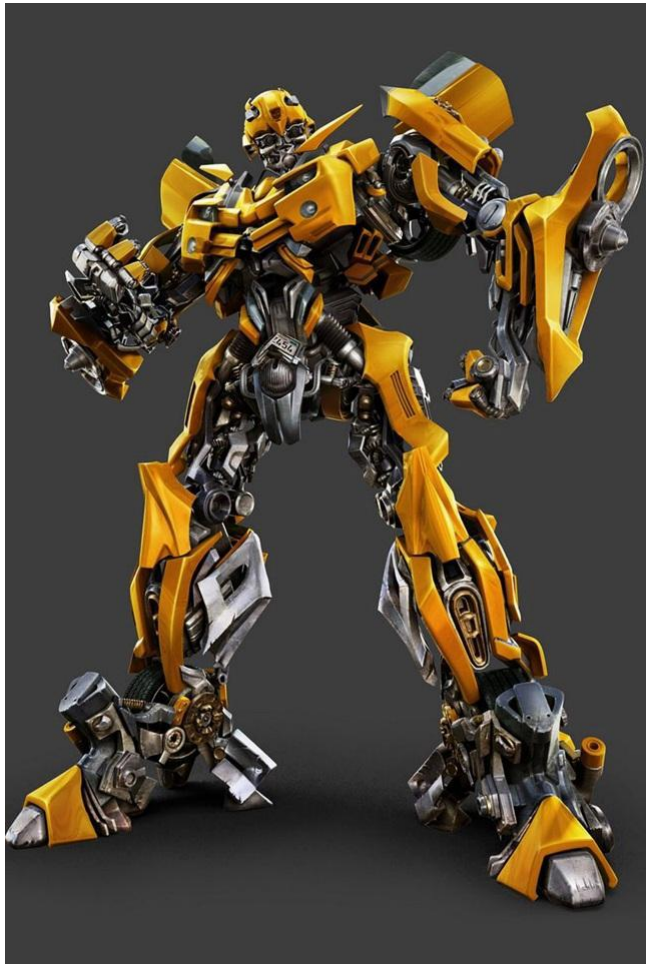
解码器层DecoderLayer：

$$d_{mid1} = LayerNorm(d_{in} + MaskedMultiHeadAttention(d_{in}))$$
$$d_{mid2} = LayerNorm(d_{mid1} + MultiHeadAttention(d_{mid1}, e_{out}))$$
$$d_{out} = LayerNorm(d_{mid2} + FFN(d_{mid2}))$$

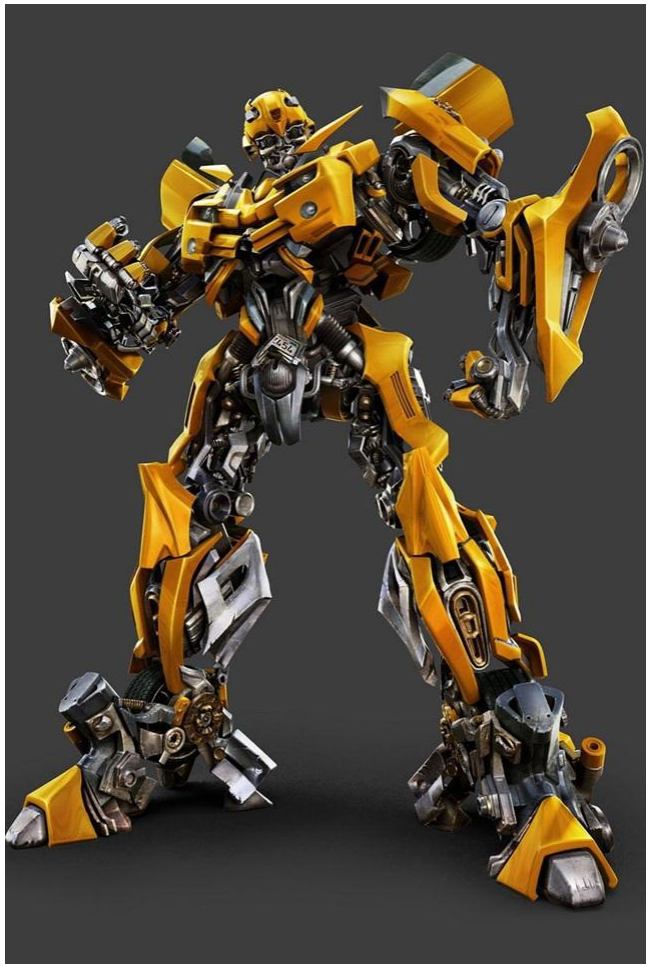
其中，$d_{in} \in \mathbb{R}^{M \times d_{model}}$为解码器层输入，$d_{out} \in \mathbb{R}^{M \times d_{model}}$为解码器层输出，$MultiHeadAttention(\cdot)$为多头注意力机制，$FFN(\cdot)$为前馈神经网络，$LayerNorm(\cdot)$为层归一化。

模型代码结构

https://shimo.im/mindmaps/DgYxW6yyrd9Gr8WQ/ 「Transformer Model Code Architecture」，可复制链接后用石墨文档 App 或小程序打开



休息



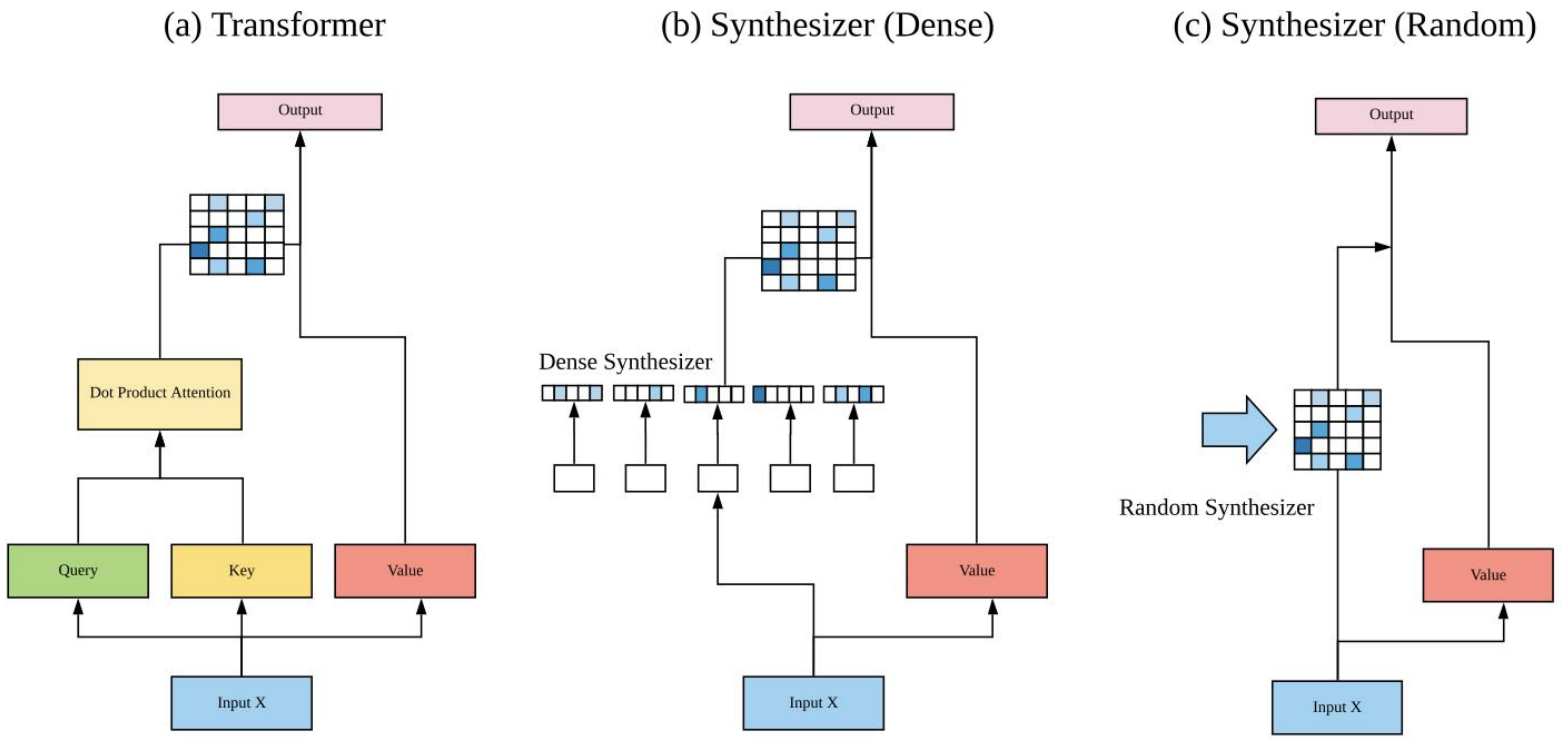
Transformer扩展

Synthesizer: Rethinking Self-Attention in Transformer Models

Reformer

Low-Rank Bottleneck in Multi-head Attention Models

Synthesizer: Rethinking Self-Attention in Transformer Models



QKV的计算方式是唯一的吗？？

Synthesizer: Rethinking Self-Attention in Transformer Models

- 自动摘要：标准注意力效果比较好
- 对话生成：标准的自注意力最差，Dense（D）和Random（R）更好
- 预训练+微调：Dense和Random迁移能力比较弱

Low-Rank Bottleneck in Multi-head Attention Models

$$\text{Attention}(\mathbf{X}) = \mathbf{W}_v\mathbf{X} \cdot \text{Softmax}\left[\frac{(\mathbf{W}_k\mathbf{X})^T(\mathbf{W}_q\mathbf{X})}{\sqrt{d_k}}\right] = \mathbf{W}_v\mathbf{X} \cdot \mathbf{P}.$$

假设输入序列长度为: n = 512
hidden_size: d = 768
头数: h = 12

Low-Rank Bottleneck: 2 * 512 * 768/12 < 512 * 512

Low-Rank Bottleneck in Multi-head Attention Models

$$\text{Attention}(\mathbf{X}) = \mathbf{W}_v\mathbf{X} \cdot \text{Softmax}\left[\frac{(\mathbf{W}_k\mathbf{X})^T(\mathbf{W}_q\mathbf{X})}{\sqrt{d_k}}\right] = \mathbf{W}_v\mathbf{X} \cdot \mathbf{P}.$$

假设输入序列长度为: n = 512
hidden_size: d = 768
头数: h = 12

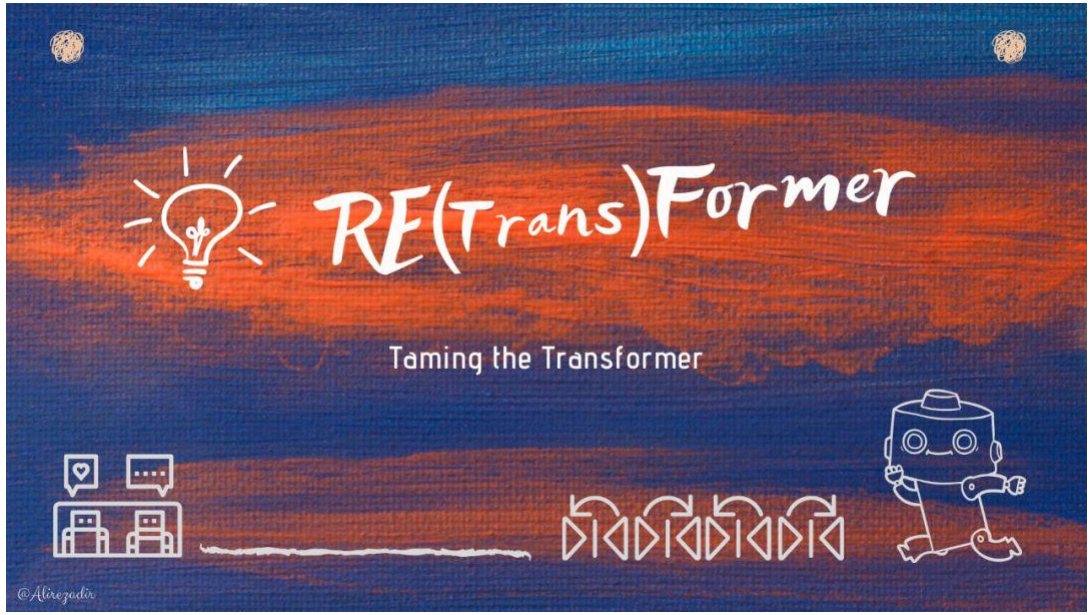
Low-Rank Bottleneck:
2*n*d/h < n*n
2 * 512 * 768/12 < 512 * 512

Low-Rank Bottleneck in Multi-head Attention Models

Low-Rank Bottleneck:
2*n*d/h < n*n
2 * 512 * 768/12 < 512 * 512

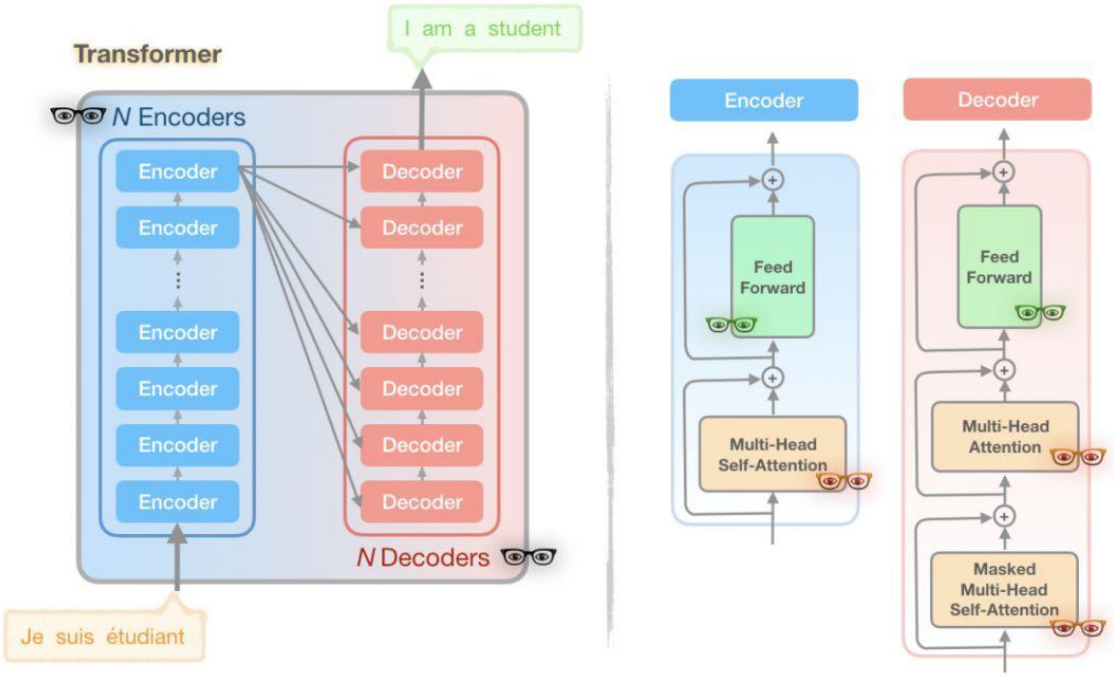
增大d/h （head_size）？
64->128
12*128 too large!!
减小h?
设计更大的key_size(QK对应的维度，V对应的不变)

Reformer



在单GPU上就可以运行的Transformer模型，而且几乎不损失精度。

Reformer

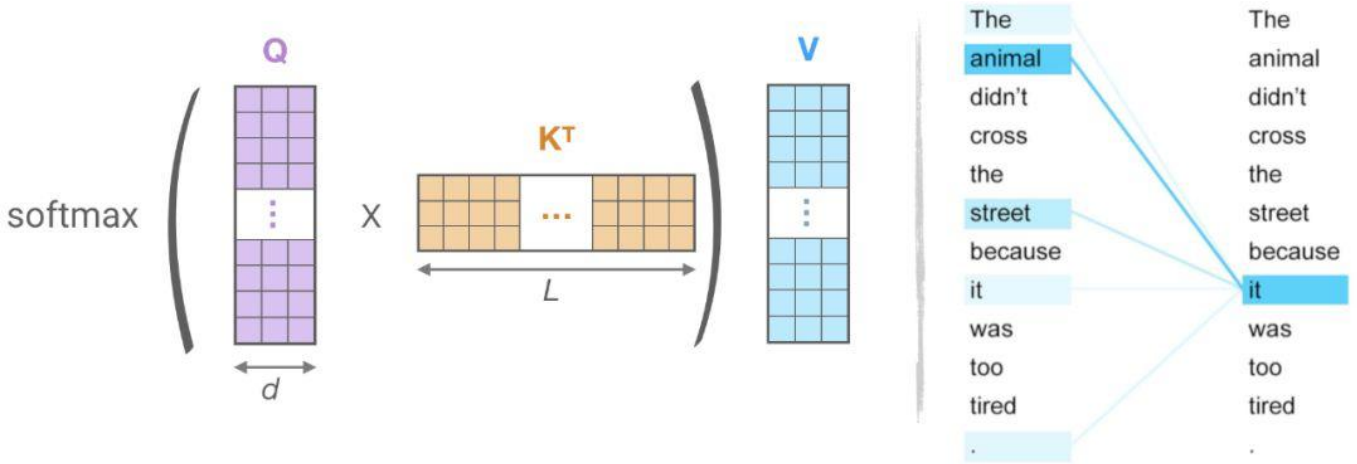


红色:自注意力计算
假设 Q、K、V： [batch size, length, d_model]
QK： [batch size, length, length]
序列长度： 64k
batch size为 1 的情况下: 如果是 32 位浮点计算就需要 16GB 的内存

黑色: 层数多
具有N层的模型要消耗N倍于单层模型的内存，因为每一层中的激活都需要存储以进行反向传播。

绿色: 前馈网络的深度

Reformer



$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Q: *L* **Q**ueries of size *d*, to attend for
K: *L* **K**eys of size *d*, to attend to
V: *L* **V**alues of size *d*

L: length of sequence
d: depth of attention

softmax*(QK^T)，由最大的元素决定的，通常是稀疏矩阵。
对于每个查询q，只需要注意最接近q的键k。

Reformer

解决思路：

- softmax更关注更大的值：对于每个 q，只需要考虑一小部分最接近的 keys
- topK -> LSH

Reformer

由于每一层的激活值需要被存储并用于反向传播，因此层数越多，占用内存越大。
一个典型的 Transformer 模型有 12 个或更多的层，因此，如果用来缓存来自每个层的值，那么内存很快就会用完。

解决思路：
可逆 Transformer，在反向传播期间按需重新计算每个层的激活值，而不是将其存储在内存中。

x -> y
y -> x

典型面试题

- 为什么要引入注意力机制？
- 什么是自注意力机制?
- 自注意力机制的计算过程？
- 什么是多头注意力机制？
- Transformer中如何计算位置向量？
- 相较于传统的机器翻译模型（rnn2rnn+attention），应用Transformer 有什么优势？

参考链接

- The Illustrated Transformer
http://jalammar.github.io/illustrated-transformer/

- 注意力用的是过滤器，而非探照灯
https://neu-reality.com/2019/10/how-brain-pays-attention/

参考链接

- Synthesizer: Rethinking Self-Attention in Transformer Models
- https://kexue.fm/archives/7430

- 《Low-Rank Bottleneck in Multi-head Attention Models》
- https://arxiv.org/pdf/2002.07028.pdf

- reformer
- https://zhuanlan.zhihu.com/p/99737208
- https://ai.googleblog.com/2020/01/reformer-efficient-transformer.html

- 可逆残差网络
- https://www.cnblogs.com/gczr/p/12181354.html
- https://spaces.ac.cn/archives/6482